UNIVERSIDAD AUTONÓMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Herramienta para la gestión de un club deportivo

Ana Isabel Viciana Pérez
Tutor: Francisco Jurado Monroy
Ponente: Rosa María Carro Salas

JULIO 2019

Herramienta para la gestión de un club deportivo

AUTOR: Ana Isabel Viciana Pérez
TUTOR: Francisco Jurado Monroy

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2019

Resumen (castellano)

Este Trabajo Fin de Grado tiene como objetivo satisfacer las necesidades de los clubes deportivos. En el día a día de un club deportivo es necesario compartir mucha información, especialmente para los eventos menos habituales, puesto que es necesario compartir el lugar, y la hora, a la que comienza el evento, a la que tienen que estar los deportistas. Así como que equipamiento deben de llevar y quienes son los que van a ir.

Las herramientas encontradas que no son específicas de ningún club, en la actualidad no solucionan estos problemas para los deportes que pueden ser practicados individualmente, como pudiera ser el tenis, el pádel, judo, gimnasia rítmica, gimnasia acrobática, y natación, entre otros muchos.

Para dar solución a este problema se ha implementado una aplicación web SPA (Single Page Application). El objetivo de esto es tener una herramienta que, de soporte a diferentes dispositivos electrónicos, que pueda ser accesible por cualquiera de los principales de un club: administrador, entrenadores, deportistas y tutores de deportistas. Una de las motivaciones para hacerla web, era que pudiera ser accesible incluso estando en la propia pista, pero que al mismo tiempo se pudiera ver en un ordenador.

Este proyecto ha sido elaborado fundamentalmente en JavaScript, uno de los lenguajes más populares de la actualidad, habiendo hecho uso de Node.js, Express y MongoDB para el lado del servidor y de Angular.js, para el lado del cliente. Las rutas del cliente se han manejado mediante el empleo de AngularUI Router. Además, se ha empleado HTML, y CSS, junto con la creación de diferentes componentes, así como la utilización de componentes de Angularjs Material siguiendo las guías de Material Design.

Abstract (English)

The aim of this Bachelor Thesis is to satisfy sport clubs necessity. In day to day, a sport club has a lot of information, specially when it is about the not usual events. That is because it is necessary to share the place and the hour, which is the planned time for starting the event, and in which one the athletes must be there. Also, it is needed to know which will be the equipment that they must have and who is going to the event.

The applications founded that are not from an specific club or sport, currently do not solve this problems for the sports that can played individually, as can be tennis, padel, judo, swimming, between others.

To give a solution to this matter we have implemented a SPA (Single Page Application) web application. The aim of this is to have an application that supports different electronic devices. Also an application that can be access by all the stuff of a club: administrator, coaches, athletes and tutors.

This Thesis has be implemented fundamentally in JavaScript, one of the most popular programming languages nowadays.

Palabras clave (castellano)

Aplicación web, SPA, componentes

Keywords (inglés)

Web application, SPA, components

Agradecimientos

Ante nada, quiero agradecer a mis padres, que siempre han estado ahí, en las buenas y en las malas. No me cabe la menor duda que si he llegado hasta aquí ha sido gracias a su apoyo.

No puedo tampoco olvidar a mi hermana, a mi familia, a mis amigos, a mis compañeros de trabajo, ni a mis alumnas y alumnos de patinaje, que cuyas risas, apoyo, y ánimos nos han hecho crecer como personas y como profesionales, y eso, realmente es invalorable y no debe ser olvidado.

Además, quiero agradecer a mi tutor, Francisco, por su apoyo en este proyecto, hasta el último momento.

INDÍCE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La motivación para realizar este Trabajo de Fin de Grado viene dada a partir de mi experiencia como entrenadora en un club deportivo de patinaje artístico. Debido a ello, he podido apreciar la necesidad existente de una aplicación la cual permitiera a entrenadores, deportistas y el resto de usuarios pertenecientes a un club deportivo, comunicarse y facilitar la organización para todos, especialmente, siendo cada vez más fácil, el acceso a la tecnología para todos los usuarios pertenecientes a un club deportivo. Incluso poco a poco los deportistas más jóvenes tienen más posibilidades de acceso a dispositivos móviles. Sin embargo, en los departamentos de administración, de empresas y asociaciones, es más habitual el uso de los ordenadores en lo que a la gestión de estas se refiere.

Puesto que se desea elaborar una aplicación que sea utilizada por todos los componentes de un club deportivo, los motivos anteriormente citados me han motivado a realizar una aplicación web que dé forma a esta herramienta. De esta manera, no se obliga a los usuarios a tener un móvil u otro dispositivo determinado para poder ver aquello que se le quiere comunicar, controlar o gestionar.

1.2 Objetivos

La gestión de un club deportivo comprende de una gran amplitud de tareas, siendo la primordial la de gestión de usuarios del club, así como los eventos deportivos y entrenamientos en los que estos participan. Así, resulta interesante contar con la gestión de la información de contacto de un deportista, tanto propia como por lo menos un contacto de emergencia. A pesar de que no sea lo más habitual precisar de este último, ante posibles lesiones, serán estrictamente necesarios en casos de gravedad. Por desgracia, disponer de ellos normalmente suele complicar la agenda de contactos de los entrenadores, antes de facilitar esta tarea que debería de poder gestionarse lo más rápido posible. Incluso se imposibilita más la gestión cuando el entrenador presente no es el entrenador habitual del deportista lesionado.

Por situaciones como las anteriormente citadas, uno de los objetivos del Trabajo de Fin de Grado es que sea accesible desde cualquier plataforma, y que no exista la necesidad de instalar nada para acceder a ello, estando accesible vía web. Además, tendrá que ser responsive y seguir unas guías de estilo para que sea sencilla de utilizar.

La siguiente gestión esencial es la de los entrenamientos, saber el lugar y la hora en la que se van a realizar, a qué deportistas va dirigido y cuáles son los entrenadores que dirigirán la sesión, así como la descripción de la misma, qué tipo de trabajo se va a realizar, etc.
La herramienta diseñada e implementada en este Trabajo de Fin de Grado busca poder satisfacer como mínimo estas necesidades básicas para deportes tanto de equipo como individuales. Con el fin de crear una aplicación que no sea específica para un deporte y que pueda ser usada por diferentes clubes.

Además de las necesidades básicas anteriormente mencionadas, se quiere disponer de un control de niveles y ejercicios, a fin de poder mejorar la evolución de los deportistas y equipos. Esto facilitará la planificación de las sesiones ayudando así al entrenador a

disponer de más tiempo para pensar más en las metodologías que quiera aplicar a cada sesión dependiendo de los deportistas que asistan a las mismas.

1.3 Organización de la memoria

El resto de la memoria consta de los siguientes capítulos:

- Capítulo 2: Estado del arte

  En este capítulo se muestra un estudio sobre las aplicaciones actuales, que dan una solución al objetivo de este Trabajo de Fin de Grado. Cabe destacar que se buscan aplicaciones multiplataforma, priorizando aquellas que no tienen necesidades específicas de instalación, como el caso de las aplicaciones web.

- Capítulo 3: Análisis

  El capítulo de análisis hace referencia a la identificación de los casos de uso y, la definición de los requisitos que tendrá nuestra herramienta para cumplir con el objetivo marcado y las herramientas que se han utilizado para el desarrollo.

- Capítulo 4: Diseño y desarrollo

  En el diseño se describe la arquitectura de la aplicación, empleando para ello, los diagramas correspondientes y explicaciones de cómo se ha desarrollado la herramienta.

- Capítulo 5: Integración, pruebas y resultados

Proporciona una explicación de las pruebas que se han realizado, y los resultados de las mismas.

- Capítulo 6: Conclusiones y trabajo futuro

Comprende las conclusiones a las que se han llegado con el Trabajo de Fin de Grado, y el trabajo que será posible realizar en un futuro.

2 Estado del arte

2.1 Aplicaciones con objetivos similares

En esta sección se detallan las aplicaciones con objetivos similares, tratando de identificar sus características y carencias.

2.1.1 GESDEP.NET

GESDEP.NET 41[1] es una aplicación web que cuenta con 20 años de experiencia, y que busca facilitar la gestión tanto administrativa como deportiva de un club deportivo. Se trata de una aplicación de pago, pero se proporciona una demo para poder visualizar la mayoría de la funcionalidad de la aplicación, la cual es muy completa. Se permite el acceso a la aplicación de diferentes integrantes del club, incluso permitiendo a los padres acceder de manera opcional y con un control de permisos asociado. En adicción se menciona que tiene la posibilidad de conectarse a la web del club, creada por la misma empresa.

Se proporciona una gran variedad de funcionalidades, como la posibilidad de gestionar fichas de usuario, donde entre otras opciones, se encuentran datos personales, estadísticas, asistencia, sanciones, estudios, expedientes disciplinarios, lesiones y control médico. A nivel de equipo, se proporciona la posibilidad de planificar con una gran variedad de ejercicios, e incluso crearlos uno mismo. Gestiona, además, sesiones de entrenamiento, pizarra táctica con movimiento, calendarios, entre otros, e incluso permite el seguimiento de jugadores de otros clubes.

A pesar de todo ello, está específicamente diseñada para los deportes: fútbol, baloncesto, fútbol sala y rugby, no contemplando la posibilidad de incluir otros deportes que puedan ser más minoritarios.
Por otro lado, no tiene una interfaz intuitiva. En el vídeo de presentación se indica que sí la tiene, especificando que tiene "*un intuitivo menú con más de 80 apartados*", pero añade a continuación que cada uno está explicado en un videotutorial y que es materia formativa en muchas escuelas de entrenadores. Desde nuestro punto de vista, esto da a entender que, para el uso de la aplicación, es necesario ver estos videotutoriales o haber realizado un curso antes.

Viendo la versión demo (ver Figura 1), observamos que hay demasiada información y carga cognitiva en la pantalla y que, además, no se adapta a su tamaño completamente, es decir, no se trata de una aplicación *responsive* capaz de adaptarse a la pantalla del dispositivo.

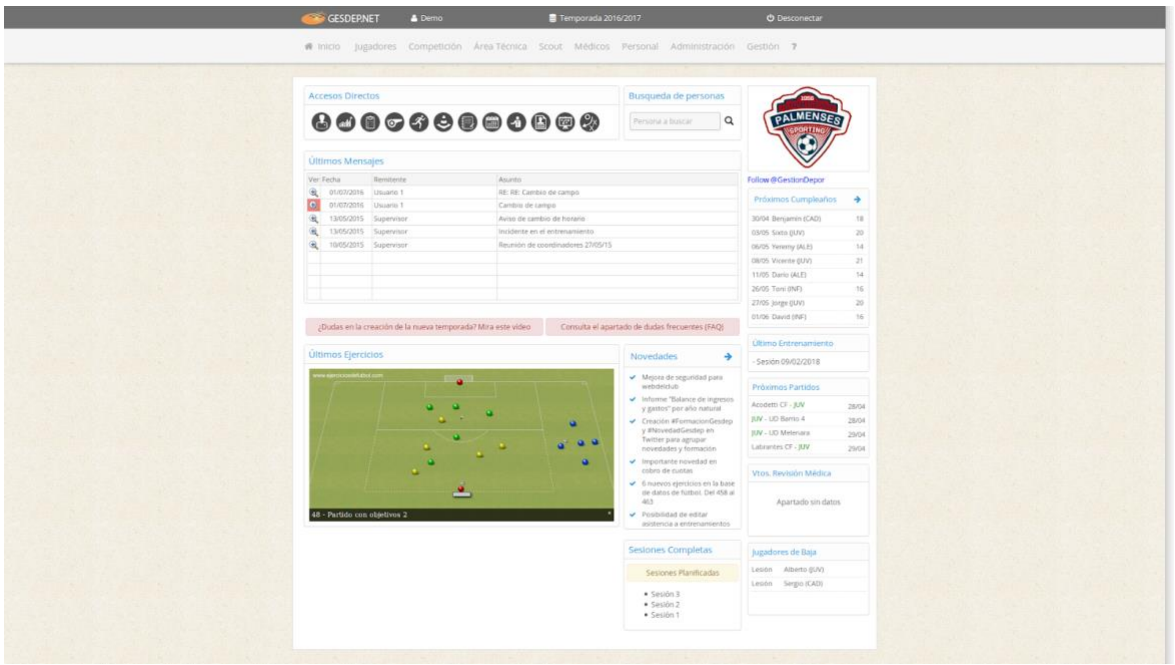


Figura 1: Aplicación GESDEP.net - Pantalla inicial versión escritorio

Para comprobar si se adapta al tamaño de otros dispositivos utilizamos la herramienta que proporciona Mozilla Firefox, comprobando que no sigue un diseño responsive. Mostrando la misma pantalla que veíamos previamente en la vista para Apple Iphone 6s según la herramienta anteriormente mencionada se ve con claridad lo mostrado en la Figura 1 (ver Figura 2).

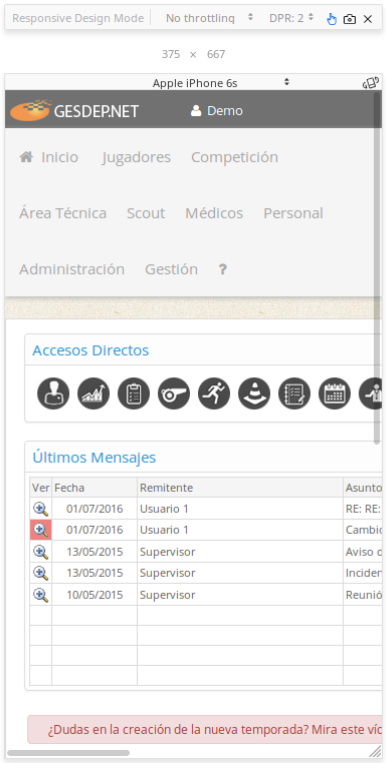


Figura 2: Aplicación GESDEP.net - Pantalla inicial versión escritorio vista desde móvil

2.1.2 Sport Easy

Sport Easy [2] es una aplicación web para la gestión de clubes deportivos que da soporte a gran variedad de deportes: waterpolo, voleibol, ultimate, street hockey, rugby, polo, lacrosse, kayak polo, hockey sobre patines, hockey sobre hielo, hockey hierba, fútbol australiano, fútbol americano, fútbol, floorball, críquet, béisbol, balonmano, baloncesto y además da la posibilidad de utilizar una modalidad "otro", para aquellos deportes que no están listados anteriormente.

Proporciona la posibilidad de personalizar la aplicación web con los colores del club, planificar eventos, partidos, entrenamientos, torneos o fiestas post partido, y enviar notificaciones por correo electrónico o móvil, para los eventos de forma que los jugadores tan solo tengan que presionar si o no. En adicción, permite la organización del uso compartido de coche geolocalizando de los conductores y los pasajeros del equipo.
Por otra parte, se pueden valorar los partidos y los jugadores, pudiendo analizar el rendimiento de los jugadores y de los equipos. Así como, se puede crear un esquema táctico, con las diferentes posiciones del equipo. Pudiendo además compartir fotos, vídeos y comentarios confidencialmente, guardándose toda la información de temporada a temporada.

Se trata de una aplicación de pago, si bien resultan gratuitas ciertas funcionalidades. En concreto, la versión gratuita, dispone de calendario, asistencia, colectas y fotos hasta 300 Mb. Es la versión de pago más cara, en la que se permite la gestión de usuarios y posibilidad de enviar mensajes y notificaciones. En la primera versión de pago se permite ver las estadísticas y gestionar a más de un equipo. Sin embargo, en ninguna de estas modalidades se da soporte a los deportes individuales.
En cuanto al diseño, esta aplicación es mucho más cómoda que la anterior (ver Figura 3).

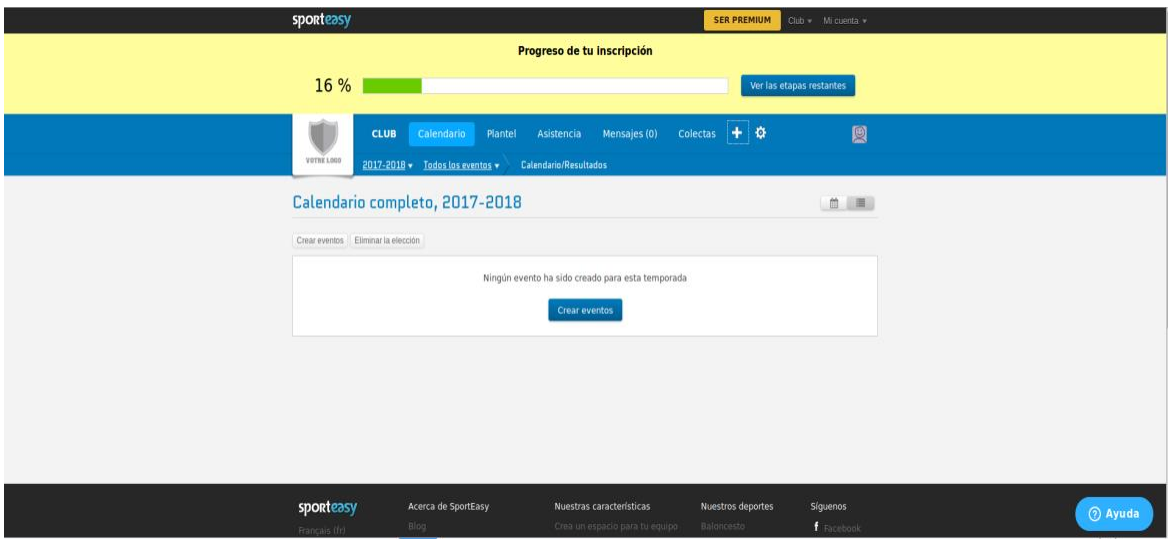


Figura 3: Aplicación Sport Easy - Pantalla de calendario versión escritorio

Sin embargo, a pesar de su comodidad, la interfaz sigue sin ser responsive, y el soporte para dispositivo móvil se proporciona mediante aplicaciones separadas, una para Android y otra para iPhone. A modo de ejemplo, la anterior pantalla en iPhone 6s queda como se muestra en la Figura 4.

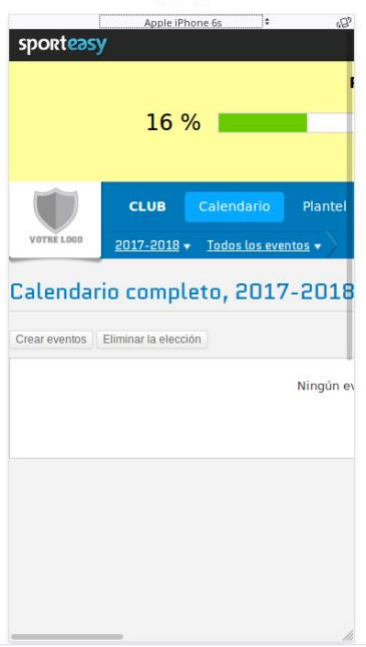


Figura 4: Aplicación Sport Easy – Pantalla de calendario versión móvil

2.1.3 Clubs TeamStuff, Coach Stuff y TeamStuff

Clubs Team Stuff, Coach Stuff y Team Stuff [3], son un conjunto de aplicaciones web separadas, pero que se dan soporte entre sí para proporcionar una funcionalidad completa a los clubes, equipos, y entrenadores deportivos.

Es un conjunto de aplicaciones que proporcionan gran variedad de funcionalidad. Sin embargo, no se termina de explicar para qué sirve cada una y cómo están integradas. Puedes estar usando una aplicación para algo que en realidad está mejor implementado y desglosado en otra de las aplicaciones, de las cuales no tienes un conocimiento inicial de su existencia. De hecho, cuando las estuve probando, no fue hasta que exploré un poco más en su página principal que encontré la existencia de una tercera aplicación.

Por tanto, desde el punto de usabilidad de los usuarios de un club deportivo, los cuales precisarán de todas ellas, se puede considerar que es bastante complejo para lo que debería de ser.

Por otra parte, no da soporte a deportes individuales, y tampoco usa un diseño responsive, aunque a diferencia de las otras, por lo menos, se encarga de ocupar toda la pantalla con un diseño a priori aceptable. Al igual que en otros casos mencionados con anterioridad, proporciona aplicaciones específicas para dar soporte a los móviles. A modo de ejemplo, puede verse una captura de la aplicación en la Figura 5 y en la Figura 6 la misma pantalla como se muestra en iPhone 6s.

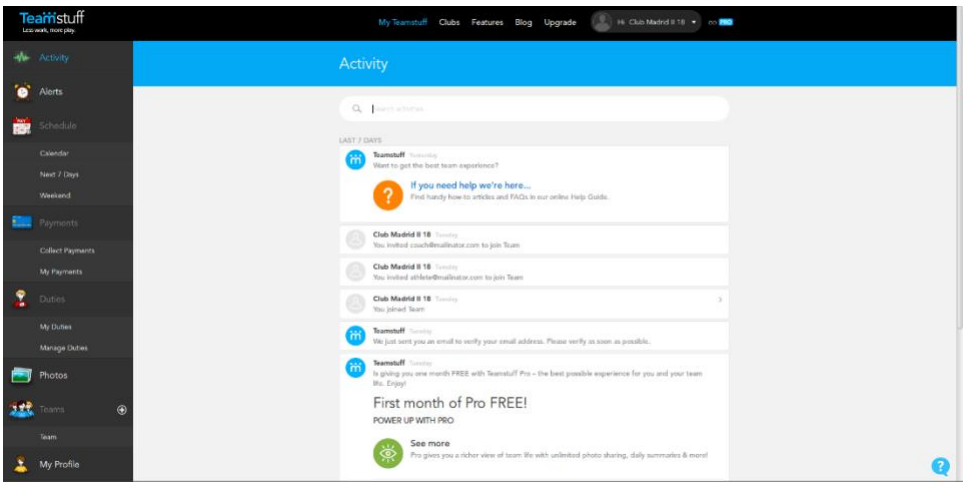


Figura 5: Aplicación TeamStuff – Pantalla inicial versión escritorio

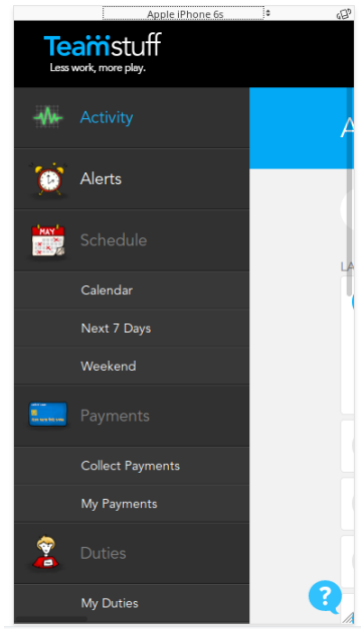


Figura 6: Aplicación TeamStuff – Pantalla inicial versión móvil

2.1.4 Comparativa entre las aplicaciones encontradas

A modo de resumen, en la Tabla 1 se muestra información relevante de las anteriores aplicaciones identificadas en el estado del arte.

En ella puede verse la carencia de alternativas que proporcionen una aplicación accesible desde diferentes dispositivos (PCs, móviles y tablets) mediante una interfaz responsive, que permita cubrir las principales necesidades de los clubes deportivos para gestión de usuarios, entrenamientos y eventos, en deportes tanto individuales como en equipo.

	Diseño responsive	Soporte deportes individuales	Soporte deportes colectivos	Intuitiva, o no sobrecargada	Gestión de usuarios
GESDEP.NET	No	No	Si	No	Si
sporteasy	No	No	Si	Si	Si (versión de pago)
Conjunto de aplicaciones Club TeamStuff	No	No	Si	Si	Si

Tabla 1: Comparativa entre las aplicaciones explicadas en el estado del arte

3 Análisis

3.1 Casos de uso

3.1.1 Definición de roles para la aplicación

En esta aplicación, se va a dar soporte a 4 roles:

- Administrador
- Entrenador
- Tutor
- Deportista

Estos cuatro roles van a tener la posibilidad de acceder a la aplicación, pudiéndose registrar únicamente el administrador del club. El resto de roles necesitarán que el administrador correspondiente a su club cree una cuenta específica para ellos asociada a su usuario, puesto que su existencia en el sistema no implica el acceso a este.

3.1.2 Casos de uso del administrador

El administrador será capaz de poder gestionar todos los módulos existentes en la aplicación, siendo este rol el encargado de crear al resto de usuarios y asignarles los correspondientes roles. Además, será quien creará las cuentas asociadas a dichos usuarios, por tanto, si un administrador no crea una cuenta asociada para un determinado usuario, ese usuario no podrá acceder al sistema.

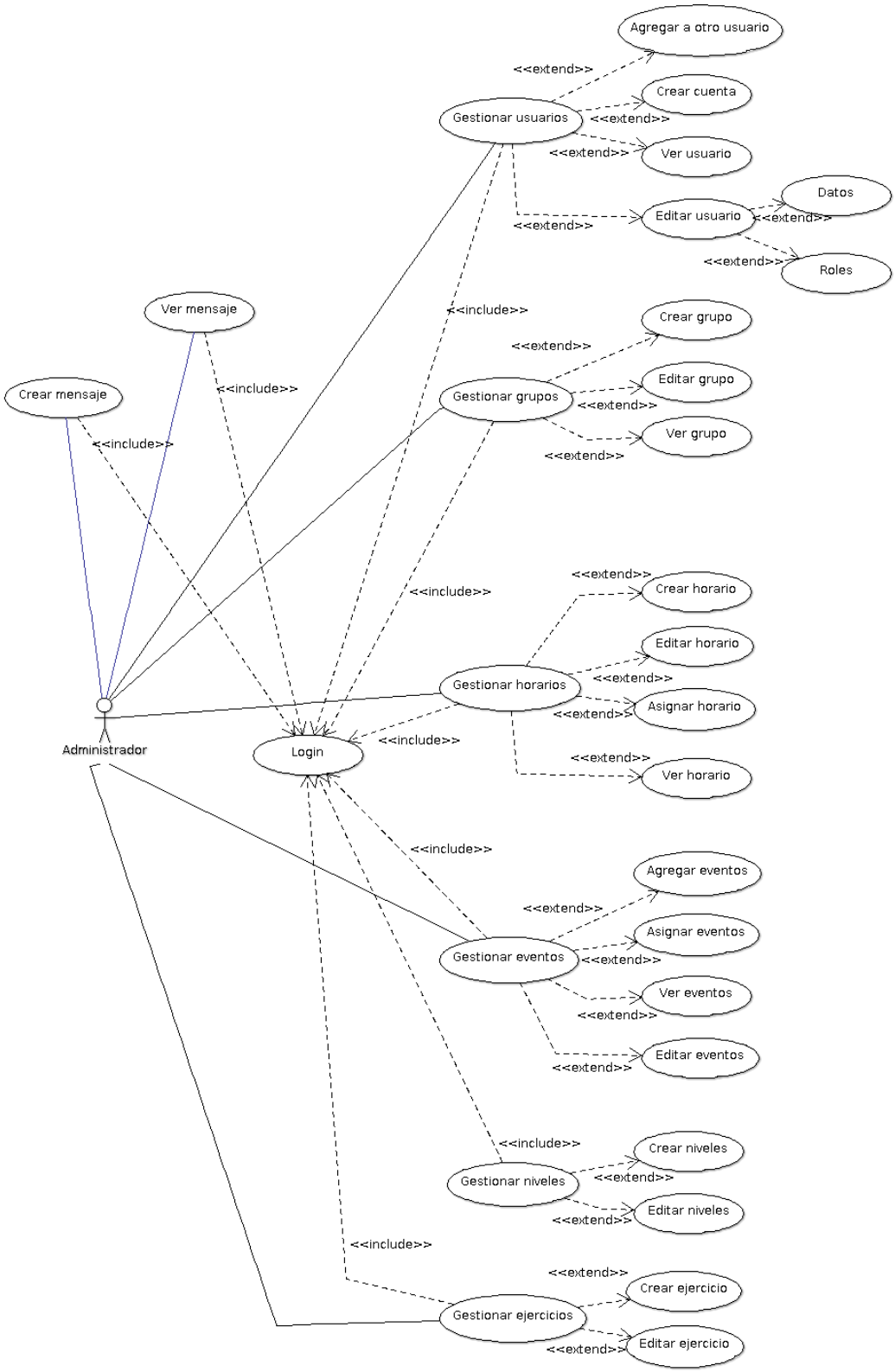


Figura 7: Casos de uso del administrador

3.1.3 Casos de uso del entrenador

El rol de entrenador es el que va a poder ver con mayor claridad toda la información de la que dispone el administrador.

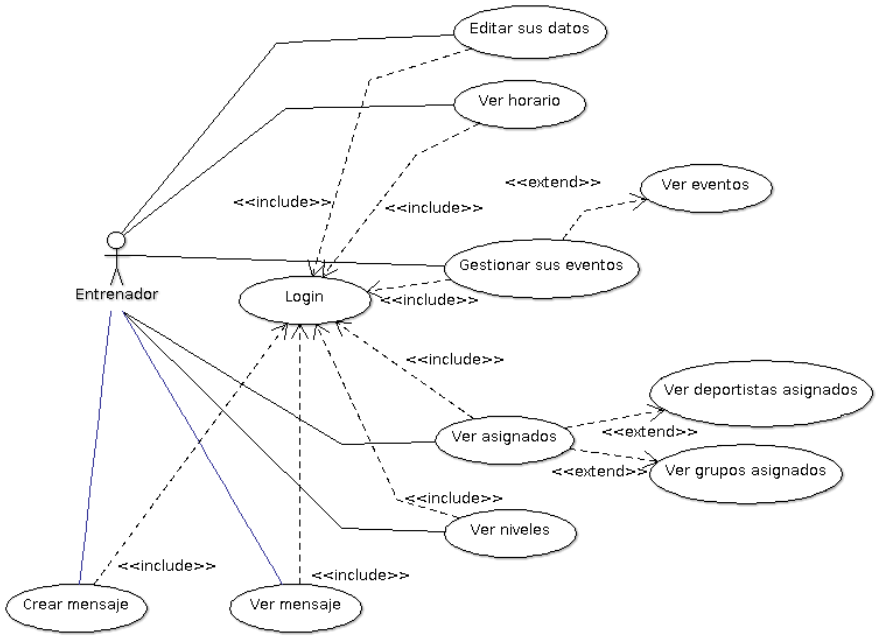


Figura 8: Casos de uso del entrenador

3.1.4 Casos de uso del tutor y el deportista

Los roles de tutores y deportistas son los que tendrán capacidades más parecidas entre ellos. Esto es debido a que los tutores podrán realizar las mismas operaciones que los deportistas que tengan asociados, además de la posibilidad de modificar sus propios datos.

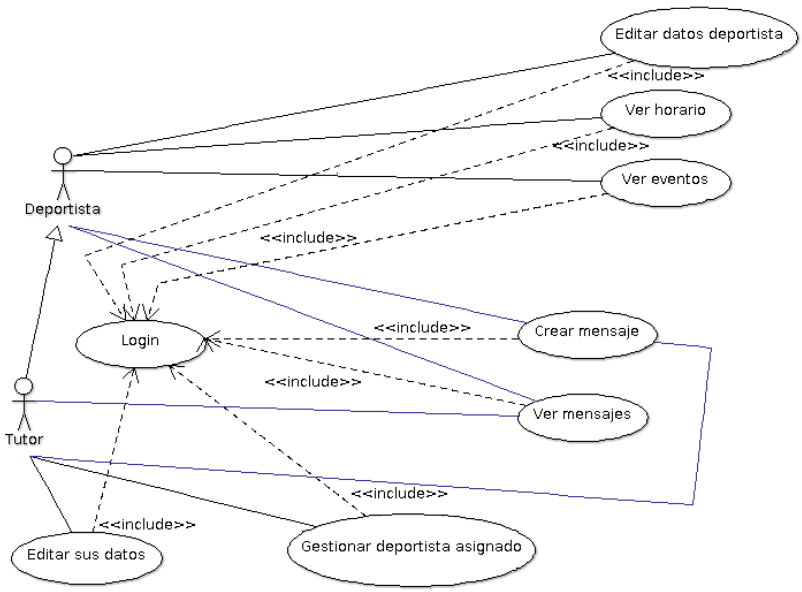


Figura 9: Casos de uso del tutor y el deportista

3.2 Requisitos

Tras la anterior definición de roles y casos de uso, en esta sección se enumeran los requisitos tanto funcionales como no funcionales.

3.2.1 Requisitos funcionales

1. Distinción entre 4 roles: administrador, entrenador, deportista y tutor.
2. Control de las fichas deportivas de los usuarios. Esto hace referencia a poder llevar un control de los datos de cada usuario, como son: nombre, apellidos, forma de contacto y observaciones médicas específicas para la actividad y generales, así como el nivel en el que se encuentra actualmente de cara a la planificación de los futuros entrenamientos, y especialmente de temporadas.
3. Permitir que un usuario tenga más un rol.
4. Control de niveles. Se podrán crear niveles que clasifiquen la intensidad y especificidad de los deportistas y grupos.
5. Facilitar la comunicación entre deportistas y entrenadores. Que los entrenadores puedan enviar mensajes a los deportistas.
6. Controlar el horario de los entrenamientos de los diferentes entrenadores y deportistas que componen el club.
7. Controlar los deportistas que hay en cada entrenamiento, así como sus entrenadores.
8. Control de los eventos deportivos. Conocer en qué eventos ha participado cada deportista y grupo, y la clasificación obtenida en los campeonatos o partidos.
9. Control de ejercicios.
    a. Se podrán crear ejercicios específicos para cada nivel creado.
    b. Se podrán añadir ejercicios a un entrenamiento.
10. Acceso a los datos de los entrenadores, deportistas y tutores
11. Edición de los datos de los entrenadores, deportistas y tutores por parte de los mismos o el administrador
12. Visión de los eventos por parte de los entrenadores, deportistas y tutores.
13. Posibilidad de envío de mensajes entre los diferentes usuarios.

3.2.2 Requisitos no funcionales

1. Que sea una aplicación accesible desde móviles, ordenadores y tablets. Una aplicación web.
2. Interfaz de usuario:
    a. Responsive: La aplicación será adaptable para poder ser usable, en ordenador, tablet o móvil.
    b. La aplicación procurará ser lo más intuitiva posible.
3. Confiable. Se ofrecerá seguridad de datos respecto a los roles, un usuario no podrá ver aquello sobre lo que no tenga permisos.
4. Seguridad. Para poder acceder a los datos deberá de estar registrado y con la sesión iniciada en el sistema.

3.3 Elección de tecnologías a emplear

3.3.1 Introducción

Como en toda aplicación web, para comenzar se debe decidir en que se van a implementar backend y frontend. Existe una gran variedad de posibilidades para cada uno de ellos, y más aún si pensamos en las combinaciones de todos. Además, necesitaremos de un gestor de paquetes para administrar todos aquellos paquetes externos que podamos necesitar, así como administrar nuestras propias versiones. Por otra parte, con el fin de añadir consistencia a la aplicación se hace necesaria una base de datos, y hay que decidir cuál usar.

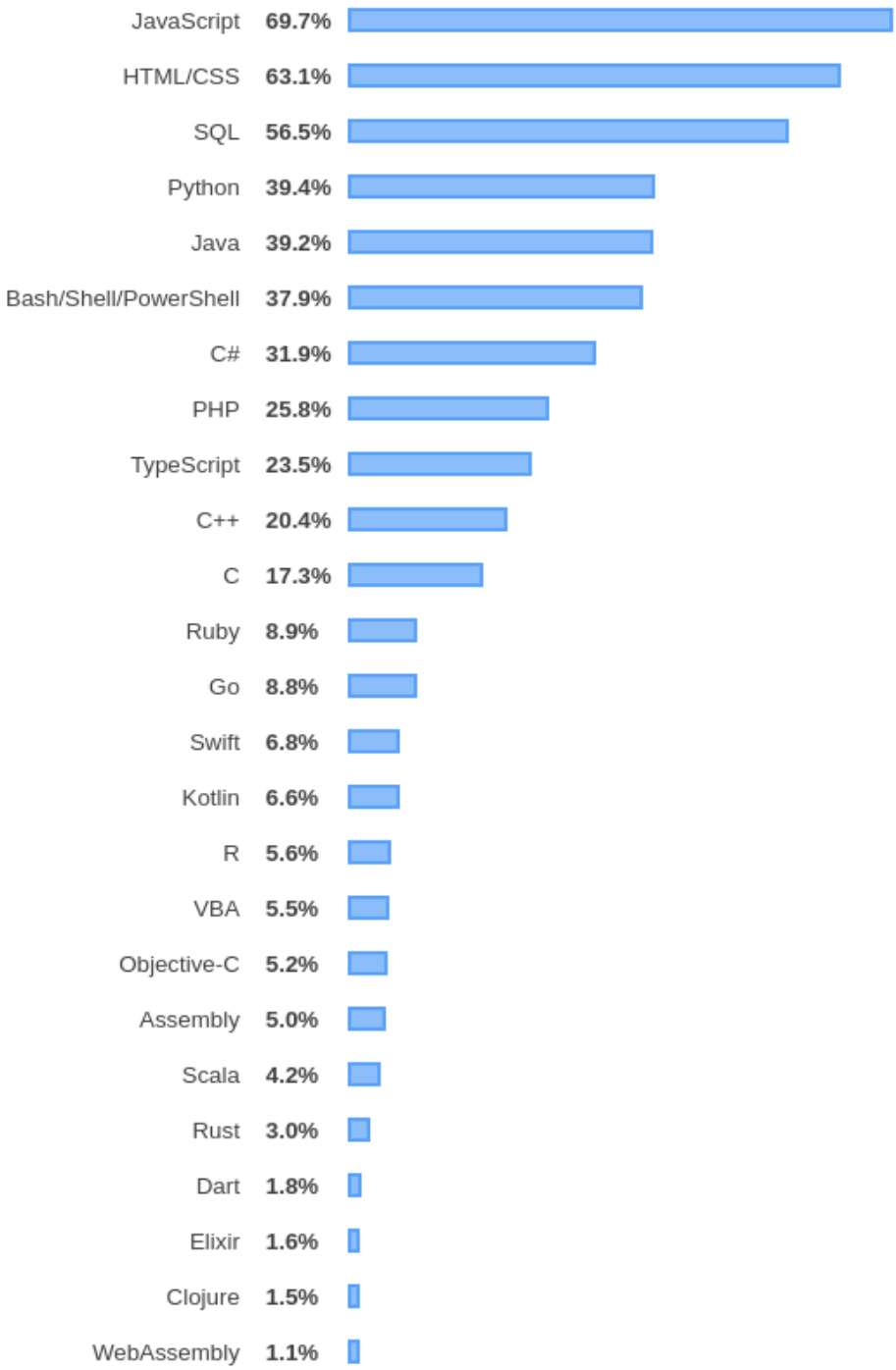


Figura 10: Resultados del sondeo de StackOverFlow de tecnologías usadas (2019)

De esta forma se hizo un estudio preliminar sobre que tecnologías utilizar, entre las distintas posibilidades estudiadas destacaban lenguajes como Java, Php y JavaScript. En la actualidad, JavaScript es el más popular entre los perfiles profesionales, como podemos observar en la encuesta de 2019 de StackOverFlow [3], una de las páginas más conocidas por los desarrolladores de todos los ámbitos (ver Figura 10).

De hecho, según los resultados de este sondeo, es el primero tanto de la lista de profesionales como del general. Por tanto, con el fin de realizar una aplicación con tecnologías recientes o que hubieran tenido una amplia comunidad, y dado que, además, se va a hacer una aplicación web dinámica, se requieran de diferentes tecnologías ligadas al lenguaje JavaScript.

Por otro lado, surgió el término, MEAN, que comenzó haciendo referencia a la posibilidad de utilizar JavaScript como full-stack, de forma que se hace posible utilizar JavaScript tanto en el lado del cliente como en el lado del servidor. En concreto, hace referencia al uso de Mongodb, Express, Angularjs y Nodejs. Aunque actualmente también se usa para referirse a la misma combinación, pero cambiando Angularjs por Angular [5] (basado en TypeScript, aunque gran parte se pueda escribir en la última versión de TypeScript). A nosotros nos interesará la opción que hace posible el uso de JavaScript en ambos lados, puesto que esto permitirá que se mantenga el código con mayor facilidad. Empresas como IBM, siguen apostando por esta solución [5].

En los siguientes apartados explicaremos cada una de ellas, así como otras tecnologías que son necesarias para el proyecto. Sin embargo, previamente, especificaremos que en concreto vamos a trabajar con ES6 (ECMAScript2015) [6], puesto que entre otros la implementación de las clases que proporciona nos será muy útil tanto para el backend como para el frontend, proporcionándonos gran legibilidad. Además, en cuanto a compatibilidad con los navegadores es cercana al 90% según se puede comprobar en caniuse [7], en concreto se indica que el porcentaje es de un 95,25%.

Más concretamente, entre las posibilidades que nos ofrece, especificaremos las que vamos a utilizar en el cliente en este Trabajo de Fin de Grado, de forma que podremos ver el porcentaje de compatibilidad que tendremos en los navegadores en la Tabla 2.

Característica	**Porcentaje de compatibilidad**
ES6 Classes	91,25%
ES6 Template Literals	91,89%
let	93,81%
Arrow functions	91,11%
Const	98,2%
String.prototype.includes	91,88%
Promises	92,7%

Tabla 2: Porcentajes de compatibilidad de las características utilizadas en la aplicación cliente

De esta manera, si observamos la Tabla 2, el mínimo porcentaje de compatibilidad que tenemos para el cliente es del 91,11%. En caso de que se viera necesaria la ampliación de este porcentaje se podrían investigar compiladores como BabelJs [9].

3.3.2 Gestor de paquetes - NPM

Entre los diferentes gestores de paquetes que existen actualmente para el lenguaje JavaScript, destacan dos, a saber: Yarn, y NPM. Existía una tercera alternativa, Bower, pero esta ha sido declarada obsoleta por sus creadores, por lo que su utilización en un proyecto nuevo, no tendría sentido [10]. Por otro lado, la que más aparece en las documentaciones consultadas en este proyecto, ha sido NPM.
Con esto finalmente, en este proyecto se decidió usar NPM.

NPM es un gestor de paquetes para Node.js, creado en el 2009 como un proyecto de código abierto para ayudar a los desarrolladores de JavaScript a compartir con facilidad los paquetes de código. Este proporciona un cliente para la línea de comandos que permite a los desarrolladores instalar, publicar y gestionar sus versiones. Además, el npm Registry es una colección pública de paquetes de código abierto para Node.js, aplicaciones web de frontend, aplicaciones móviles, robots y routers entre otros [11].

Permite diferenciar entre instalar paquetes en nuestro disco y en nuestro proyecto de forma sencilla mediante flags. Si se trata de paquetes que precisan del uso de la línea de comandos hay que tener en cuenta que, si se instalan únicamente en el proyecto, solo serán accesibles desde los scripts de dicho proyecto. Un ejemplo de la clase de paquetes a los que nos referimos serían aquellos dedicados a los tests.
Para controlar las versiones de los paquetes utilizados en el proyecto, utilizamos un fichero llamado "*package.json"* que se encuentra en la raíz del mismo. En este fichero, se indica toda la configuración del proyecto: nombre, versión, descripción, scripts, dependencias, dependencias de desarrollo y otras. Existen varios paquetes que, de hecho, precisan y/o permiten añadir su configuración en este fichero.
Destaquemos el hecho de que en este fichero contiene la versión de nuestro proyecto. NPM no solo nos permite saber con facilidad las versiones que estamos usando de otros paquetes, sino que además nos permite gestionar nuestra propia versión. Podemos subir de versión con tan solo el comando "*npm version*", seguido de que a cual se quiere subir, o si es una subida de major, minor, patch u otro, basta con escribir la opción correspondiente a cada una de ellas, pues también lo admite [12].

3.3.3 MongoDB

Para la implementación de la base de datos se ha utilizado MongoDB. Se trata de una base de datos no relacional, que guarda los datos en documentos. Estos documentos son guardados con una notación muy similar a un JSON (JavaScript Object Notation) y los representa mediante con la codificación binaria de los mismos, llamada BSON [13].

3.3.4 Express

El framework inspirado en Sinatra y más conocido de node.js es sin duda Express y tambien es la librería subyacente para un gran número de otros framework de node populares.

Express está basado en JavaScript y proporciona una delgada capa de características de aplicación web básicas sin dejar de lado todas las características de Node.js.
Entre otras características, ofrece Router de URL (GET, POST, PUT...), lo que sirve de utilidad para motores de plantillas (Jade, EJS, JinJs, etc.), Middleware vía connect y una gran capacidad de alcance de test [14].
Express es una estructura sencilla y muy útil para realizar aplicaciones. Se pueden definir los módulos que utilizaremos para nuestra aplicación, por ejemplo; express, routes, http, pat. Con esto gestionamos las rutas de una aplicación asociando cada ruta con una función encargada de controlar la acción a ejecutar.
Entre sus ventajas encontramos que Express no es dogmático o transigente, por lo que se puede insertar cualquier middleware compatible.

3.3.5 Angularjs

AngularJs es un framework de desarrollo para JavaScript creado por Google, cuya finalidad es facilitar el desarrollo de aplicaciones web que sean Single Page Application. Entre sus propósitos se encuentra la separación entre el frontend y el backend y nos proporciona herramientas para trabajar con los elementos necesarios para la creación de un cliente web de una manera sencilla y óptima [15].

3.3.6 Node.js

Node.js es un servidor en JavaScript basado en una arquitectura orientada a eventos, utilizando un modelo asíncrono y dirigido por eventos.
Fue creado por Ryan Dahl en febrero de 2009, poseyendo la influencia de varios sistemas entre los principales tenemos Even Machine y Twisted [16].

3.3.7 Momentjs

Para el apartado de calendario, se hace necesario un gestor de fechas. Entre las posibles opciones que se podían utilizar, destacaban DateJs y MomentJs. Sin embargo, a pesar de que DateJs tenía buenas posibilidades, tiene una mantenibilidad muy baja por parte de sus mantenedores, y por otro lado, MomentJs es ampliamente utilizado, e incluso se usa en AngularJs Material, anteriormente citado.
MomentJs que permite la utilización de fechas teniendo en cuenta el horario local con el que se ha configurado, así como el idioma. Además, proporciona funcionalidad para formatear las fechas según se quieran mostrar, así como sumar y restar fechas, entre sus muchas características. [17]

3.3.8 Angularjs Material

AngularJs Material es un marco de componentes de UI y una implementación de referencia de la especificación de Material Design de Google. Proporciona un conjunto de componentes UI reutilizables, bien probados y accesibles basados en Material Design. Esta herramienta, es la clave para conseguir mayor usabilidad puesto que emplea las guías de estilo que el usuario ya puede conocer de otras aplicaciones [18].

3.3.9 Tests - Jest

Existen diferentes opciones a la hora de testear JavaScript: Mocha, Jasmine y Jest, son las más destacadas. Siendo Jest la más popular entre ellas, a pesar de que Mocha es más antigua y por tanto tiene más madurez, sin embargo, quizá la principal diferencia entre

ellas es que Mocha precisa de más configuración para funcionar. Por tanto, se ha decidido utilizar Jest, dado que además está basado en Jasmine [19].

3.3.10 Calidad del código - JSHint

En lenguajes no tipados como JavaScript, se hace necesario el uso de alguna herramienta que nos indique cualquier posible mala utilización del código, esto es, un linter [20]. Entre los linters disponibles, se ha elegido el programa JSHint, un linter con una sencilla aplicación que nos permite configurar diferentes reglas para tenerlas en cuenta en la salida de su ejecución, aunque en este Trabajo de Fin de Grado se han dejado las de por defecto [21].

3.4 Plataformas y herramientas utilizadas en el desarrollo

3.4.1 Sublime Text

Editor de textos avanzado, preparado para trabajar con diferentes lenguajes de programación. Permite además la incorporación de extensiones que faciliten el trabajo de los programadores. Entre sus múltiples características, cabe destacar que permite la edición de múltiples líneas, a diferencia de otros editores más comunes [22].

3.4.2 GetIt

Programa para GNU/Linux que permite el envío de peticiones HTTP a endpoints de una API REST y muestra la respuesta dada por el servidor permitiendo, además, el envío de peticiones GET y POST personalizados [23].

4 Diseño y desarrollo

4.1 Detalles de implementación de la Base de datos

4.1.1 Introducción

Para el desarrollo de la persistencia de los datos en MongoDB ha sido necesario tener en cuenta una serie de decisiones que han influido en el desarrollo.
En primer lugar, había que decidir con que driver se iba a implementar la aplicación, así como cuál iba a ser el diseño de la base de datos.

4.1.2 Driver

Para desarrollar la base de datos, se ha utilizado el driver oficial de MongoDB, con el cual nos encontrábamos con una inconsistencia, y es que había que hacer N instancias para N peticiones, y a su vez, cerrarlas todas ellas. Por tanto, se iba a repetir el mismo código en tantas ocasiones como veces que se necesitara consultar a la base de datos, o realizar alguna petición de modificación a la misma. Lo cual, no solamente es posible motivo de error, sino que, además, dificulta las correcciones o la incorporación de nuevos requisitos en las conexiones a la base de datos.

Por otra parte, cuando se requiere de la concatenación de peticiones a la base de datos, el código se podría hacer bastante ilegible y con ello poco mantenible, a causa de todas las consultas anidadas.
Por todos estos motivos, se implementó un módulo aparte dedicado a gestionar toda la parte de persistencia donde:

1. La petición que se realizara tenía que estar dentro de un callback.
2. La conexión con la base de datos debía cerrarse con cada petición, algo complicado dado que NodeJs sea asíncrono, lo cual dificultaba la concatenación de peticiones a la base de datos.

Finalmente, a la solución a la que se llegó pasa por el uso de funciones delegadas, de la forma que se muestra en la Tabla 3.

```
const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');
const logger = require('../logger');
// Connection URL
const url = 'mongodb://localhost:27017';
// Database Name
const dbName = 'sportclub';
const databaseConnection = databaseUse => {
  MongoClient.connect(url, (err, client) => {
    assert.equal(null, err);
    logger.info('Connected to database');
    const db = client.db(dbName);
    databaseUse(db, () => {
      logger.info('End connection to database');
      client.close();
    });
  });
```

```
};
module.exports = {databaseConnection};
```

Tabla 3: Código de dataConnection.js

En la figura puede verse una función que recibe como parámetro la función *databaseUse*, la cual será la que realizará las operaciones que precise con el cliente. Para que pueda hacer uso de este cliente, se lo pasamos como primer parámetro. Una vez que este termine de utilizar la base de datos correspondiente, debe llamar al callback que se le ha pasado como segundo parámetro, en el cual nosotros cerramos esa conexión con la base de datos, solucionando así el problema con el que nos habíamos encontrado.

4.1.3 Diseño de la base de datos

El diseño inicial ha sufrido varias modificaciones durante el desarrollo del proyecto, puesto que tenemos una gran variedad de datos. Para comenzar tenemos la colección "*users"*, que es aquella que almacena todo aquello que común para todos los usuarios, como es el nombre, primer apellido, segundo apellido, teléfono de contacto, teléfono de emergencia y observaciones.
Al principio de este diseño, en esta clase también había incluido el username y la contraseña de la cuenta asociada ellos. Sin embargo, el administrador no es un usuario como tal y no se precisa incluir todos esos datos, por tanto, esta información (username y contraseña de la cuenta para acceder al sistema) se extrajo a otra colección, llamada "*account"*.
Por otra parte, surgía el problema de cómo se registra por primera vez el administrador. Por lo que se añadió otra colección llamada "*club"*, que poseía el nombre del club correspondiente. De esta manera, se incorporó en todas las colecciones "raíz" el identificador del club y esto hace que en la actualidad se permita que la aplicación sea usada por clubs diferentes, sin que estos sean conscientes de la existencia de otros excepto a la hora de elegir el nombre de usuario de la cuenta, puesto que no se permiten repeticiones.
Hasta aquí, hemos mencionado 3 colecciones, *club*, *account* y *users*.
Por otro lado, tenemos las colecciones *athlete*, *coach* y *tutor*, que corresponden a la información de deportistas, entrenadores y tutores, respectivamente. Cada una de ellas guarda la información específica de cada tipo usuario, como son los deportistas asociados para el tutor, o el nivel del deportista, o los niveles que imparte el entrenador. Los identificadores de estas colecciones son almacenados en la colección *users*, de forma que la información queda enlazada un usuario.
Existe una colección específica para los niveles, "*level"*, que son generados por el administrador, que guarda el nombre y la dificultad de estos, así como una descripción.
Otra colección nos permite crear un ejercicio, con un nombre, una descripción y una dificultad. Un ejercicio se puede enlazar con un nivel, guardándose, el id del nivel en el documento correspondiente la colección "*exercise"*.
En adicción, para dar soporte a los mensajes se ha creado una colección "*message"*, que guarda el identificador de la cuenta desde la que se envía el mensaje, y el identificador de la cuenta al que se le envía, la fecha y si ha sido consultado, además del mensaje.
Para el soporte de los grupos se ha creado una colección, llamada "*groups"*, que contiene los identificadores de los deportistas asociados a él.
La mayor complejidad se ha dado a la hora de elaborar el calendario, puesto que se quería reducir el número de apariciones en la base de datos y un deportista usualmente, puede tener hasta 4 entrenamientos diferentes en una semana. Si a esto le añadimos que puede

haber unas 10 personas implicadas en un entrenamiento, podemos pensar en que habrá 40 instancias. Sin embargo, hay 2 posibilidades para almacenar estos datos:

- Repetir el entrenamiento por cada deportista. Por lo que, por cada 10 usuarios en un entrenamiento, habría que repetir 10 veces el documento del entrenamiento.
- Repetir el deportista por cada entrenamiento. Por lo que, cada usuario estará 4 veces repetido en el ejemplo propuesto.

Con el fin de no perjudicar a ninguno de los usuarios elegido la segunda opción, que además será la que más favorezca al administrador, puesto que le interesará el conjunto del club y no un solo usuario.

4.2 Arquitectura de la aplicación

La arquitectura de la aplicación sigue un esquema cliente-servidor empleando un patrón Modelo-Vista-Controlador.
Para desarrollar el lado del servidor o backend de la aplicación se ha implementado un API REST. Lo cual, mediante el uso de Express, se ha podido desarrollar con facilidad. Además, se ha elaborado una Single Page Application.
La principal característica de este tipo de implementación es que el servidor no almacena datos de sesión de ningún tipo y todo el acceso a la capa de dominio y persistencia de la aplicación se gestiona mediante peticiones GET y/o POST.
Por su parte el lado del cliente o frontend, ha sido implementado como una aplicación web dinámica empleando JavaScript siguiendo las directrices de Material Design para el diseño de la interfaz y conseguir con ello una interfaz responsive y usable.

Como se ha mencionado, para conectar el cliente y el servidor, para el desarrollo de la aplicación se ha llevado a seguido un patrón Modelo – Vista – Controlador implementado del siguiente modo:

En el lado del servidor, disponíamos del modelo y del controlador, siendo el controlador el que se encargaba de hacer las peticiones a la base de datos pasándole el modelo. En la Tabla 4 puede verse un ejemplo del código de una petición de registro en el lado del servidor.

```
baseRouter.post('/register/club', (req, res) => {
    const { clubName, username, pass } = req.query;
    // Encrypt password
    const salt = bcrypt.genSaltSync(10);
    const hash = bcrypt.hashSync(pass, salt);

    const newClub = new Club();
    newClub.setClubName(clubName);

    const createClub = (db, databaseCallback) => {
    const newClubData = new ClubData(db);
    newClubData.create(newClub.getClub())
        .then(resultClub => {
            const clubId = resultClub.insertedId;
            const newAccount = new Account();
            newAccount.setAccount({username, password: hash, clubId});
            newAccount.setAccountIsAdmin(true);

            const newAccountData = new AccountData(db);
```

```
            newAccountData.findAccount(newAccount.getAccountUsername())
                .toArray((err, accounts) => {
                    if(err || accounts.length > 0){
                        logger.error(err, ' accounts.length ',accounts.length > 0);
                        res.status(500).send();
                        databaseCallback();
                    } else {
                        newAccountData.create(newAccount.getAccount())
                            .then(resultAccount => {
                                const accountId = resultAccount.insertedId;
                                logger.info('clubId '+ clubId + ' accountId ' + accountId);
                                res.status(201).send({clubId, accountId});
                                databaseCallback();
                            }).catch(error => {
                                logger.error(error);
                                res.status(500).send();
                                databaseCallback();
                            });
                    }
                });
        }).catch(error => { logger.error(error); });
    };
    databaseConnection(createClub);
});
```

Tabla 4: Código de la petición de creación de un club

En la primera parte de la función, obtenemos los parámetros de la consulta y los imprimimos en donde se está ejecutando el servidor. A continuación, se encripta la contraseña para almacenarla encriptada en la base de datos.

A continuación, es cuando se ve aplicada la separación del modelo y la base de datos, siendo estos manejados por el controlador. Primeramente, creamos una nueva instancia del modelo *Club* y a continuación la rellenamos con los datos que nos han pasado como parámetros de la consulta. Acto seguido, creamos la función *createClub* que en la que usaremos la base de datos, tal y como se especifica en el anterior apartado de la memoria.

En la función *createClub* creamos una instancia de la clase ClubData. En este proyecto, en las construcciones de las clases ***Data, se selecciona la colección, en este caso en concreto 'club' y se almacena en una variable interna de la clase, ****Collection*, en este caso *clubCollection*. De esta forma la clase mantiene la colección en la que se van a realizar los cambios relativos a esa colección.

Continuando con el código, vemos que se pasa la instancia del modelo que anteriormente habíamos creado a la instancia de la colección correspondiente, concretamente a una función *create*. Esta función, inserta en la base de datos el documento que le ha sido pasado por parámetro.

4.3 Middleware

4.3.1 Introducción

El middleware en nuestra aplicación se encuentra en el lado del servidor y se llama cada vez que se realiza una petición al servidor.
Para nuestra aplicación se han introducido varias funciones en el middleware, mediante el uso de Express. Con Express se permite diferenciar entre las diversas rutas para realizar funciones diferentes dependiendo de cuál sea la que se esté solicitando, sin dejar de lado la posibilidad de ejecutar funciones para cualquier llamada al servidor.

4.3.2 Logging

En un servidor es muy importante el uso de algún sistema que permita realizar un seguimiento de las peticiones que se han realizado para así poder ver y corregir lo antes posible los errores que puedan haber surgido, especialmente en la etapa de desarrollo en la cual se dan con más facilidad caídas del sistema. Esto ocurre especialmente con Nodejs dada la falta de variables tipadas en JavaScript y a pesar de usar un linter, en este caso JSHint, no se comprueba la existencia de funciones pertenecientes a diferentes clases.Por todo ello, se ha incorporado una función al middleware que permite el logging de las aplicaciones, mediante Morgan.

Morgan es un logger para las peticiones de HTTP para node.js que permite diferentes formatos y opciones. Posee de formatos predefinidos, pero también permite usar un formato custom. Entre las opciones posibles se permite el escape, el guardar en un fichero, y escribir la línea en la solicitud en lugar de en la respuesta. [24]
Además de este Morgan, se ha utilizado Winston. Winston es un logger más general que permite parsear la salida dividiendola en varios niveles, como el de info, o error.[25]
Winston permite distinguir entre los diferentes niveles para almacenarlas en ficheros, de esta forma, en este Trabajo de Fin de Grado tenemos un fichero '*error.log'* en el que se almacenan todas las salidas con los errores, y un fichero '*combined.log'* con la combinación de los diferentes niveles, con las peticiones que se han realizado y los logs empleados en los diferentes módulos, en este caso hemos usado info y error. [25]
Para hacer uso de esta configuración, y que fuera común a todos los módulos, se ha creado en la raíz del proyecto un fichero '*logger.js*', que era importado en los diferentes módulos.

4.3.3 Parseo de las peticiones

Todos los parámetros de una petición dependen del usuario. Por ello en primera instancia no son confiables y precisan de una primera validación, para eso usamos *body-parser*, el cual extrae toda la parte del cuerpo y la expone en req.body. En nuestro caso hemos usado la opción de analizar el texto como json, mediante *bodyParser.json()*, dado que todas las peticiones esperadas actualmente precisan de este caso. [26]

4.3.4 Utilización de routers

Tener todas rutas de la aplicación en un solo módulo simplemente cuando estas exceden unas 10, puede hacer que el código no sea muy mantenible, especialmente si tenemos escrita toda la funcionalidad de los accesos a la base de datos en este mismo módulo. Por

otro lado, en la mayoría de ocasiones se podrán distinguir varios tipos de rutas, como podría ser las de relativas a usuarios y las relativas a niveles en nuestro caso, ya no solo por la lógica de las mismas, sino que normalmente se seguirá una convención para que las rutas de usuarios, siempre comiencen por "/users" y las de niveles por "/levels".
Express proporciona una herramienta que nos permite solventar este problema, separando las rutas de diferentes ámbitos. Esta herramienta se llama Router y se puede utilizar con tan solo crear una instancia de *express.Router()* y utilizar dicha instancia como si fuera la propia de Express. Para utilizar este Router dentro de la aplicación, simplemente hay que incluir un middleware en el que se especifique la ruta que va a tener y pasarle la instancia del router empleado.
En este Trabajo de Fin de Grado se ha ido un paso más allá y con el objetivo de tener bien separado lo que es la parte de configuración común y el resto de rutas, se ha creado un módulo *routes.js* que contiene una variable con una lista de objetos con la estructura {url, router}. Por ejemplo, para las rutas base (login, registro y logout) el objeto de la lista correspondiente es el siguiente:

```
{ url: '/', router: baseRouter }
```

Con la especificación de esta lista exportada se nos permite obtenerla en el módulo de configuración de forma que podamos incorporar tantas rutas como se precisen sin la necesidad de tocar este módulo clave para nuestra aplicación de la siguiente forma:

```
// Adding routes support
routes.map( route => {
    app.use(route.url, route.router);
});
```

Tabla 5: Código de la incorporación de rutas al router

4.4 Registro, login y logout

En cualquier aplicación, es necesario algún tipo de encriptado para guardar la contraseña en la base de datos. En este caso se ha utilizado el paquete *bcrypt* para esta tarea. Este paquete permite crear un hash de una cadena, con la misma facilidad, que permite contrastar el hash devuelto anteriormente con otras cadenas.
Utilizando este paquete, en la fase de registro de nuestra aplicación, lo primero que hacemos es obtener el hash de la cadena correspondiente a la contraseña. Se hacen las comprobaciones correspondientes, como que no haya otra cuenta con el mismo nombre de usuario, y se guarda en la base de datos. En la fase de login, se busca la cuenta con el username introducido, y se obtiene el hash almacenado en la base de datos de este. Una vez obtenido se contrasta con la contraseña introducida mediante el paquete ya mencionado.
Para elaborar este apartado, la mayor complicación fue como diferenciar entre cuando un usuario estaba logueado y cuando no, especialmente siendo una SPA. Para ello, cuando se contrastan la contraseña y el hash, en el caso de que coincidan, se crea una cookie, con los datos de la cuenta y del club correspondiente que son devueltos en la petición.
De esta manera, se puede mantener la aplicación abierta en múltiples pestañas y durante el tiempo que se desee en la aplicación. Al ser algo adjunto al navegador nos permite tener distintas sesiones en navegadores diferentes. Además, con esta aproximación, cada vez que un usuario ha realizado el logout, e intenta acceder a una de las páginas para las que se necesita el login, se le redirige a la página de login, sin permitirle el acceso a la aplicación.
Cuando el usuario hace logout esta cookie se borra.
Para el manejo de cookies se ha utilizado el servicio de AngularJs, $cookies.

4.5 Desarrollo del servidor

4.5.1 Creación de un usuario

Para la creación de un usuario se quería permitir la posibilidad de que pudiera tener distintos roles, es decir, que un usuario pudiera ser a su vez entrenador y tutor, por ejemplo, o tutor y deportista, o deportista y entrenador. El único rol que no se quería que tuviera posibilidad de combinación era el administrador, pues posee una serie de características que los demás no tienen, como no poseer de nombre y apellidos, además de más permisos.
Sin embargo, no se quería dejar de lado la posibilidad de que solo fuera uno de ellos, especialmente siendo este el caso más común.
Esto hacía que se necesitara poder acceder a las 3 colecciones, *athlete*, *tutor* y *coach*, sin necesidad de que estas fueran obligatorias. La dificultad de esto se hallaba en que, recordemos de nuevo, Node.js, es asíncrono y, además, MongoDB, devuelve los resultados mediante callbacks. Por tanto, en una primera instancia no se esperaban unas peticiones a otras, y concatenarlas no era la solución, puesto que esto obligaba a que el usuario tuviera los 3 roles, o que se generaran deportistas, tutores o entrenadores vacíos en la base de datos, lo cual no tenía sentido. Si nos pararamos a pensarlo por un momento, en un caso como este, para 3 usuarios se generarían 9 documentos entre las 3 colecciones de nuestra base de datos, creciendo exponencialmente a medida que creciera el número de usuarios. Ralentizar las búsquedas y generar posibles errores no era una solución válida.

La solución se encontró mediante el uso de promesas, y una de las funciones de ES6 que permiten el combinar el uso de varias promesas, *Promise.all* .
Como breve introducción al uso de promesas, comparémoslas con una función que recibe dos callback, uno para cuando se ha acabado la ejecución de la función con errores y otro para cuando todo ha salido correctamente. El callback recibido para los errores, lo identificaremos con el catch, mientras que el callback recibido para cuando todo ha sido correcto, lo identificaremos con el then. Es importante saber que se pueden concatenar callbacks, dado que en estos callbacks cuando son llamados así se devuelve otra promesa.

```
if(req.query.isCoach === 'true'){
    const { coach } = req.query;
    const newCoach = new Coach();
    newCoach.setCoach(JSON.parse(coach));
    const newCoachData = new CoachData(db);
    coachPromise = newCoachData.create(newCoach.getCoach())
        .then(resultCoach => {
                coachId = resultCoach.insertedId;
                logger.info(`introduced coach with id: ${coachId}`);
                return { coachId };
        });
} else {
    coachPromise = Promise.resolve();
}
Promise.all([athletePromise, tutorPromise, coachPromise])
    .then(promiseAllResult => {
        logger.info(`Promise.all promiseAllResult ${JSON.stringify(promiseAllResult)}`);
        logger.info(` athleteId ${athleteId}`);
        logger.info(` tutorId ${tutorId}`);
        logger.info(` coachId ${coachId}`);
```

```
        const rolesId = {};
        promiseAllResult.map(promiseResult => {
        if(promiseResult && promiseResult !== undefined){
            if(promiseResult.athleteId && promiseResult.athleteId !== undefined){
                rolesId.athleteId = promiseResult.athleteId;
            }
            if(promiseResult.tutorId && promiseResult.tutorId !== undefined){
                rolesId.tutorId = promiseResult.tutorId;
            }
            if(promiseResult.coachId && promiseResult.coachId !== undefined){
                rolesId.coachId = promiseResult.coachId;
            }
        }
    });
```

Tabla 6: Fragmento de código de la petición de creación de un usuario (parte de adicción de roles)

Si observamos el fragmento de código de la Tabla 6, vemos que hay una primera parte que hace referencia a la creación del entrenador. Hay varias variables que han sido definidas previamente en este controlador, athletePromise, athleteId, tutorPromise, tutorId y coachPromise, coachId; siendo su scope únicamente esta función, pero debido a que estamos usando JavaScript, si estas se modifican dentro de los callbacks y su valor también se ve modificado en la función superior. Si el usuario de la petición es un entrenador, nos fijamos que se almacena en coachPromise la promesa generada al introducir en la base de datos al entrenador, y que, además, se está utilizando un "callback" para que cuando todo ha sido correcto se almacene en coachId, el id del documento del entrenador que acaba de introducirse con éxito.
A continuación, nos fijamos en que cuando el usuario a crear no es un entrenador, estamos inicializando coachPromise a Promise.resolve. Promise.resolve lo que hace es devolver una promesa vacía que ha sido ejecutada con éxito, de esta manera, vamos a tener en cualquiera de los posibles casos una promesa.
Este mismo procedimiento se ha realizado para los deportistas y tutores previamente en el código.
Ahora tenemos para todos los casos una promesa, y es cuando usamos la función Promise.all. Promise.all lo que hace es esperar a que todas las promesas pasadas como lista, se resuelvan correctamente y devuelve una promesa con el resultado. A continuación, esos identificadores serán guardados en la cuenta del usuario. De esta forma, hemos conseguido poder crear un usuario con uno o múltiples roles, sin necesidad de distinguir entre todos los posibles casos.

4.5.2 Obtención de la información de un usuario

De la misma manera que en el anterior apartado, mencionábamos que MongoDB devuelve los resultados mediante callbacks, cabe destacar una diferencia entre las operaciones de creación y obtención de documentos: la información relativa a la creación de un documento es proporcionada mediante promesas, mientras que la obtención de los mismos es proporcionada mediante un método “toArray” proporcionado como resultado de la función de búsqueda.

Por tanto, para las operaciones de obtención de datos, no nos servía la metodología empleada en el anterior apartado. Además, dado que los deportistas, entrenadores y tutores tienen ids relativos a otras colecciones, complicaba la lectura del código añadir las peticiones a las colecciones necesarias para que el usuario pudiera tener esa información sin conocer la estructura de la aplicación.
Para dar solución a ello, se han creado distintas funciones de utilidad, dentro del apartado de controlador, con la siguiente declaración:

```
getAthleteData(db, athletePromise, athleteCallback)
```

Donde 'db' es el cursor de la base de datos, 'athletePromise' es el resultado de la operación de búsqueda, y 'athleteCallback' es la función que se llamará los resultados obtenidos.
Por otra parte, como no todos los usuarios tienen los tres roles, cuando se obtiene el usuario se hace una comprobación de que rol tiene, declarando 'athletePromise' como o el resultado de la obtención de la función de búsqueda de nuestra clase de datos, o la siguiente función que hemos creado específicamente para poder dar solución a esta concatenación:

```
noFindResults = { toArray: callback => callback(null, [])}
```

4.5.3 Uso de funciones reutilizables

Para la implementación del dominio, dado, que se ha buscado la reutilización del código en los modelos. Para la implementación de los setters, se ha usado la convención mostrada en la Tabla 7.

```
setLevel({clubId, name, description, difficulty, _id}){
    const currentLevel = this.getLevel();
    this.level = {
        levelId: _id || currentLevel.levelId,
        clubId: clubId || currentLevel.clubId,
        name: name || currentLevel.name,
        description: description || currentLevel.description,
        difficulty: difficulty || currentLevel.difficulty,
    };
}
```

Tabla 7: Código de la modificación de un elemento de la clase Level

De esta forma, sean cuales sean los atributos que se quieran cambiar, podemos llamar a la misma función, pasándole como objeto los atributos a cambiar, dado que se respetan los valores anteriores del objeto.
Sin embargo, para la implementación de la clase de usuarios concretamente, se ha utilizado un formato algo diferente al explicado anteriormente, dado que eran muchas más propiedades y el código se hacía poco legible en comparación. El formato al que estamos haciendo referencia se encuentra en la Tabla 8.

```
editUser(userModified){
    allowedFields.map(field => {
        if(this.checkField(userModified, field)){
            if(field !== 'userid' && userModified){
                this.user[field] = userModified[field] || this.user[field];
            }
        }
    });
}
```

Tabla 8: Código de la modificación de un elemento de la clase User

En este caso, tenemos una lista con las propiedades que se pueden cambiar en esta función, llamada allowedFields. En esta función lo que estamos haciendo es recorrer la lista de campos permitidos, y para cada uno, comprobar si es válido el introducido el objeto userModified. En caso de que sea válido y el campo no sea el id, asignamos el nuevo valor, y si no existe, dejamos el valor anterior.

4.5.4 Filtros de búsqueda

Una de las motivaciones para realizar este Trabajo de Fin de Grado era que cuando hubiera una urgencia en la pista deportiva, ya fuera evento o entrenamiento, cualquier entrenador pudiera acceder fácilmente a la información de contacto del deportista implicado. Por ello, se vio de gran utilidad la creación de filtros, especialmente uno que permitiera la búsqueda por nombre, o cualquiera de los apellidos de un usuario. A continuación, explicamos en concreto el filtro de búsqueda.

Para la búsqueda que hemos mencionado, no se han tenido en cuenta los deportistas que pudieran estar asociados al entrenador, y se busca en todo el club las personas que contengan la palabra buscada.

Con el objetivo de llegar a ello, se ha realizado una serie de funciones comunes a todas las secciones, que permiten obtener el objeto de consulta esperado por MongoDB, de forma que así quedaba mucho más legible, y además seguíamos manteniendo la separación entre los modelos, el controlador de las peticiones, y los controladores de las peticiones a la base de datos. A modo de ejemplo de cómo se han empleado estas funciones, vemos la Tabla 9, donde *User* es la clase modelo de los documentos de la colección "*users*" y *userData* es la clase de acceso a los documentos de dicha colección.

```
const clubIdQuery = obtainQueryFieldWithValue(User.CLUBID_FIELD, clubId);
const nameQuery = obtainQueryFieldContainsAString(User.NAME_FIELD,
stringToEvaluate);
const surname1Query = obtainQueryFieldContainsAString(User.SURNAME1_FIELD,
stringToEvaluate);
const surname2Query = obtainQueryFieldContainsAString(User.SURNAME2_FIELD,
stringToEvaluate);
const searchQuery = obtainQueryForSeveralConditionsOr([nameQuery,
```

```
surname1Query, surname2Query]);
const query = obtainQueryForSeveralConditionsAnd([clubIdQuery, searchQuery]);

userData.getUsersByQuery(query).toArray((err, users) => {
```

Tabla 9: Código de elaboración de la consulta de búsqueda de usuarios a partir de un literal

4.6 Desarrollo del cliente

4.6.1 Uso de componentes

Actualmente, para el desarrollo del frontend de las aplicaciones se está llevando a cabo el uso de la componetización.
La utilización de componentes nos permite la reutilización del software, puesto que, por ejemplo, si tenemos unos estilos que se repiten 3 veces en la misma o en diferentes pantallas, se puede construir un componente que se encargue de dar esos estilos, pasándole por parámetro aquello en lo que se diferencie, como será el texto a mostrar. Una vez implementado, se puede llamar tantas veces como se desee, permitiendo la simplificación del mantenimiento del sistema y las pruebas de este, dado que, para 3 elementos, tan solo necesita ser modificado una vez. Esto a su vez permite obtener una mayor calidad, puesto que el tiempo ganado se puede emplear en mejorar la funcionalidad del componente.
En nuestra aplicación hemos utilizado varios componentes, propios y los proporcionados por Angularjs Material, que a su vez nos permitían tener mayor facilidad para crear el diseño responsive que habíamos marcado como requisito.

En este Trabajo de Fin de Grado la vista está completamente basada en componentes, y esto ha ayudado a la distinción entre un usuario logueado y uno sin loguear. En la Figura 11 vemos una vista de escritorio de la aplicación, en la que se han distinguido diferentes partes.

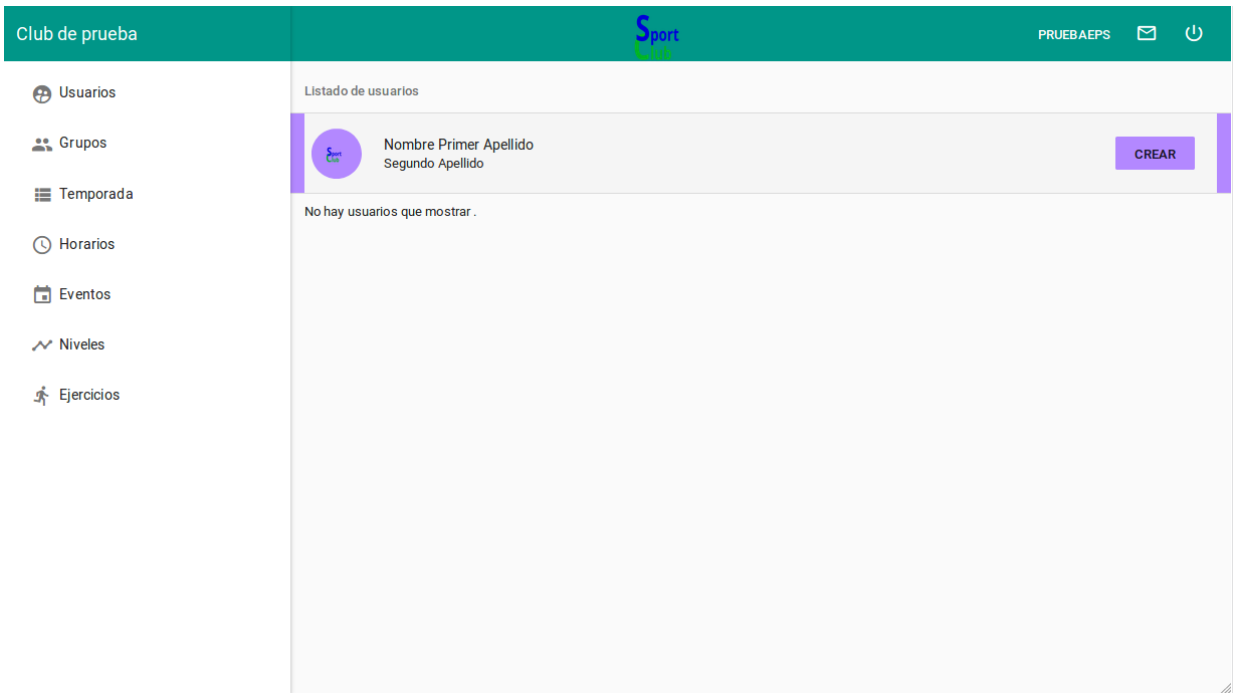


Figura 11: Pantalla de usuarios versión escritorio

- Un panel lateral, en el que se muestra el menú.
- Una barra superior en la que se muestra el nombre de la aplicación, el nombre de usuario, el botón de mensajes y el botón de salida.
- Un apartado que dependerá de la opción marcada en el menú lateral.

Todas estas partes se han controlado mediante el uso de AngularJs y los servicios que proporciona junto con la utilización de AngularUI Router. Además, se han utilizado los servicios de layouting que proporciona AngularJs Material, y sus directivas basadas en Flexbox para proporcionar una vista responsive de la aplicación, como se puede apreciar constrastando la Figura 12 y Figura 12.

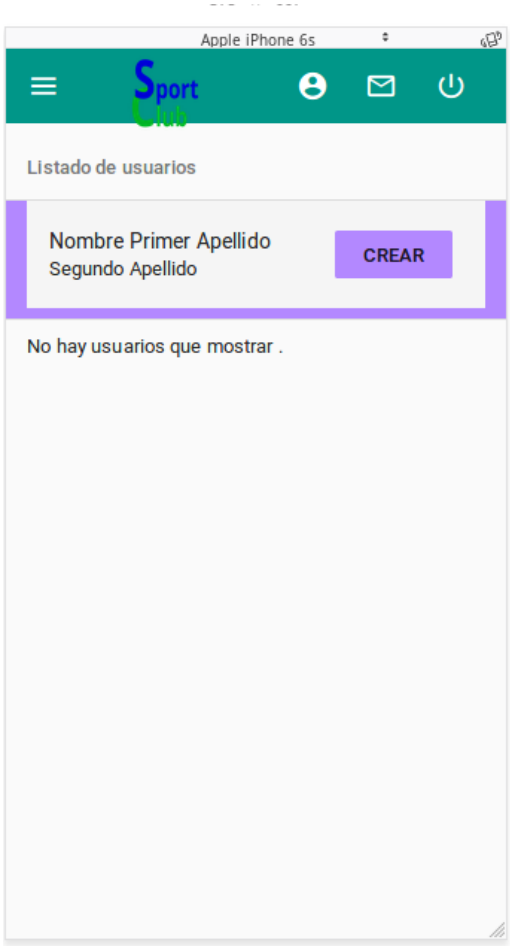


Figura 12: Pantalla de usuarios versión móvil

Si vemos la figura 18, apreciamos que el nombre de usuario ahora se ha convertido en un icono, y el panel lateral ha desaparecido, apareciendo en su lugar un botón en la barra superior. Este botón se haya en un componente propio que contiene la lógica que llama al servicio que despliega el menú lateral, como se muestra en la Figura 13.

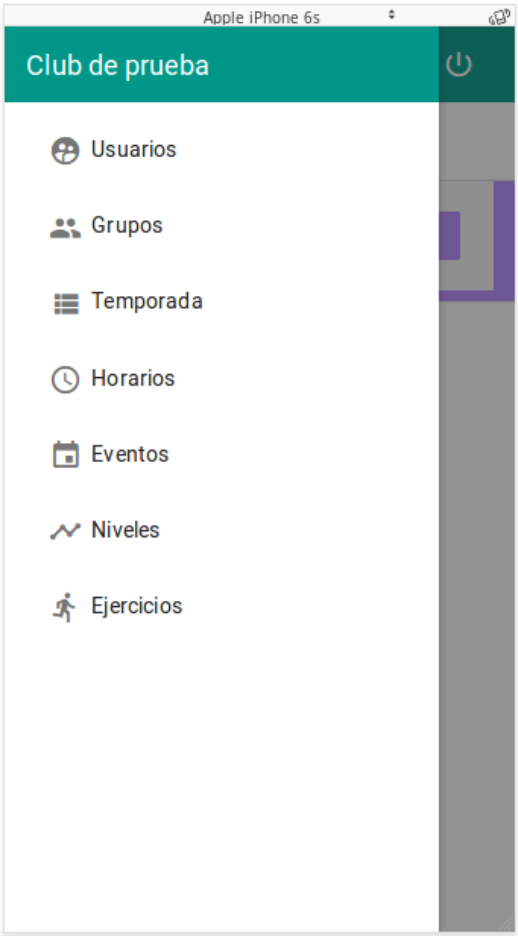


Figura 13: Menú de la aplicación versión móvil

Como podemos ver la división en componentes proporciona mucha flexibilidad en el momento de creación de nuevas pantallas.

4.6.2 Uso de componentes reutilizables

En este Trabajo de Fin de Grado nos hemos concentrado en las posibilidades de reutilización de código en los diferentes apartados de la aplicación, de forma que tanto para el servidor como el cliente se ha implementado lógica que sirve para este cometido.
En el caso del cliente, esto se ha dado en varias ocasiones.
En primer lugar, en el menú lateral. Esto se debe a que tenemos usuarios con distintos roles, por lo que las opciones laterales serán diferentes en función de ello. Por tanto, era necesaria una forma de controlar las acciones a las que puede acceder cada usuario. En nuestro caso, esto se ha implementado creando una enumeración, options con objetos que siguen una estructura determinada, por ejemplo, el objeto relativo al apartado de ejercicios, es el siguiente: `LEVELS: { name: 'Niveles', icon: 'timeline', state: 'sc.levels'}`

Tenemos una propiedad name, que será la que se muestre en el menú, junto con el icono para el cual la propiedad icon guarda el nombre del icono correspondiente del paquete de Material Icons. Acto seguido tenemos el nombre del estado que al que se va a transitar, mostrándose en la parte principal de la pantalla.

Para decidir cuales mostrar, se ha dejado que en la petición que se hace inicialmente al servidor con toda la información relativa al servidor, nos llegue una lista con las opciones que tiene ese usuario, cuyos elementos son los nombre clave del objeto *options* mencionado. En el ejemplo dado, uno de los elementos de la lista de opciones mandada por el servidor, será *'LEVELS'*.
Por otra parte, se ha creado un componente para mostrar las listas, dado que, por ejemplo, tanto el apartado de usuarios como el del niveles o mensajes sigue una estructura muy similar, a excepción de los literales que se muestran, como se puede apreciar en la Figura 14.

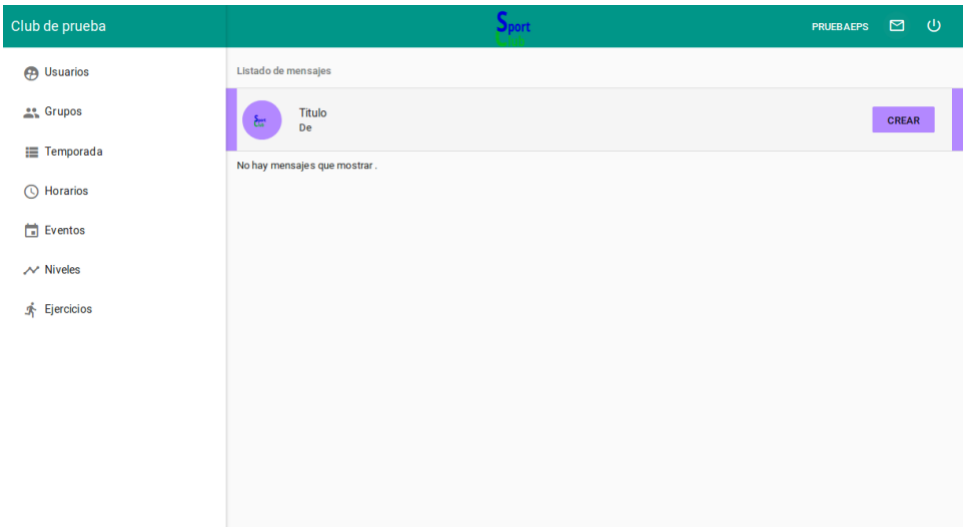


Figura 14: Pantalla de mensajes versión escritorio

Pero sin duda, una de las componetizaciones más difíciles en cuanto a lógica ha sido la de calendario. Puesto también se deseaba que tuviera un diseño responsive. En esta aplicación nos hemos valido del componente de AngularJs Material md-grid-list, para crear la estructura de columnas, sin embargo, es necesario rellenar cada una de ellas e indicar cuanto espacio se quiere que ocupe cada una. Además, este componente tiene bastantes problemas en cuanto es la interacción con el resto de componentes de la pantalla, por lo que ha sido necesario incluir también la parte que indica los días de la semana dentro de la lista del mes, dado que sino se superponía. Aun así, esto también ha tenido que ser solucionado en pantallas más pequeñas. Podemos ver el contraste entre la Figura 15 y la Figura 16.

Figura 15: Pantalla de calendario versión escritorio

Figura 16: Pantalla de calendario versión tablet – Vista de junio 2018

Este componente hace uso de MomentJs, para obtener los días del mes actual, así como los nombres de los días de la semana. Haciendo uso de las funciones que proporciona para realizar todo el cálculo de fechas que aquí se precisa, pues con el mismo componente se pintan todos y cada uno de los meses, siempre teniendo en cuenta el día de la semana en el que se comienza y el último día del mes. A modo de ejemplo, vemos también la vista de mayo que se proporciona.

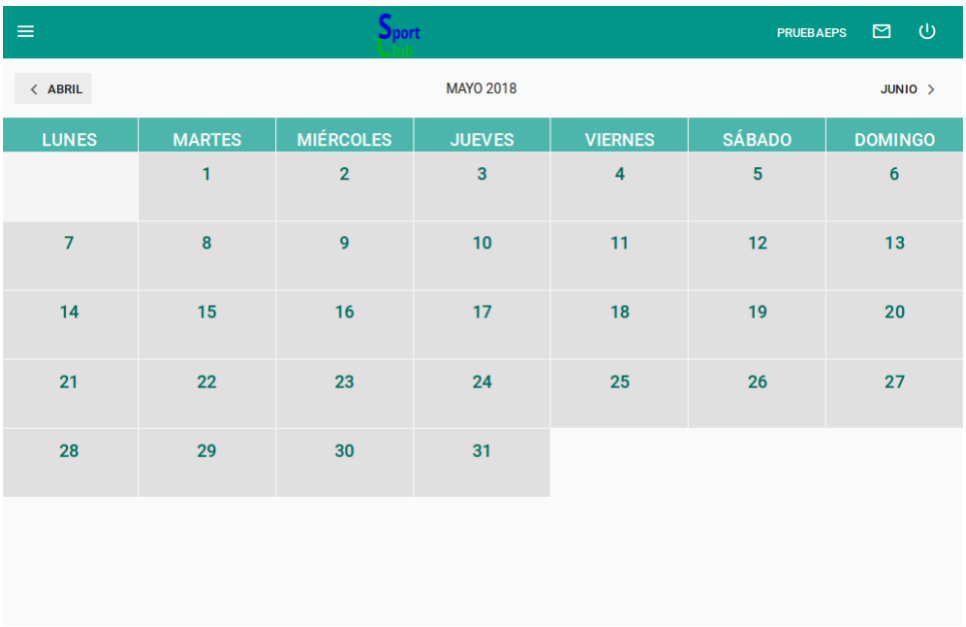


Figura 17: Pantalla de calendario versión tablet – Vista de mayo 2018

Para conseguir esto, se ha elaborado una lista con objetos como el siguiente:

```
{
    header:(i+1),
    rowspan: 2,
    rowspanSmall: 3,
    date: dayToAdd,
    seasonId: this.seasonId,
}
```

Tabla 10: Objeto de creación de los elementos de un calendario

Donde header es el número que se muestra, label, es para aquellos en los que haya algún evento se muestre el texto de esa cadena, rowspan es para indicar el tamaño de la fila por defecto, rowspanSmall es para el tamaño de las filas en pantallas pequeñas y date y seasonId, son los parámetros para ver el detalle del evento cuando se pulsa uno de los días

5 Integración, pruebas y resultados

5.1 Pruebas

5.1.1 Linter – JSHint

Linter es una herramienta que permite el análisis estático del código. De esta forma, se detectan posibles errores sin la necesidad de la ejecución del código. Esto en lenguajes no tipados como JavaScript resulta casi imprescindible en la mayoría de ocasiones. Por ello, existen múltiples herramientas que cumplen con esta función.
En particular, en este proyecto, se ha elegido JSHint para cumplir con esta función.
Con el fin de utilizarlo con mayor regularidad, se ha añadido a los comandos que inician tanto la aplicación como los tests (ver Figura 18) para que el análisis se realice de forma automática y reporte posibles inconsistencias, bugs o errores cuanto antes.

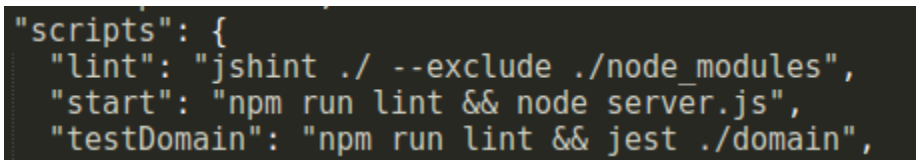


Figura 18: Fragmento de los scripts del package.json de la aplicación

Esto ha permitido identificar a tiempo múltiples errores, los cuales, de no ser así, podrían haber llegado a ser inapreciables a primera vista, pero claves para la ejecución de la aplicación.

5.1.2 Pruebas unitarias - Jest

Las pruebas de caja blanca son aquellas que se realizan conociendo la estructura de la aplicación o el código a probar. En nuestro caso, las pruebas se han realizado sobre el dominio, utilizando Jest. Jest es la solución planteada por el grupo JavaScript de Facebook para realizar pruebas en JavaScript.
Como ejemplo de las pruebas efectuadas, en la Tabla 11 vemos el test de una de las clases:

Fragmento del código que se está probando	`setExercise({clubId, levelId, name, description, performance, _id}){`
	`  const currentExercise = this.getExercise();`
	`  this.exercise = {`
	`  exerciseId: _id || currentExercise.exerciseId,`
	`  levelId: levelId || currentExercise.levelId,`
	`  clubId: clubId || currentExercise.clubId,`
	`  name: name || currentExercise.name,`
	`  description: description || currentExercise.description,`
	`  performance: performance || currentExercise.performance,`
	`  };`
	`}`

Fragmento del test realizado a dicho fragmento de código	`test('Exercise set - levelId options passed', () => {`
	`  const exerciseToSet = new Exercise();`
	`  const exerciseParams = {`
	`  levelId: 'levelId',`
	`  };`
	`  exerciseToSet.setExercise(exerciseParams);`
	`  expect(exerciseToSet.exercise.exerciseId).toBe('');`
	`  expect(exerciseToSet.exercise.levelId).toBe('levelId');`
	`  expect(exerciseToSet.exercise.clubId).toBe('');`
	`  expect(exerciseToSet.exercise.name).toBe('');`
	`  expect(exerciseToSet.exercise.description).toBe('');`
	`  expect(exerciseToSet.exercise.performance).toBe(0);`
	`});`

Tabla 11: Código de una función y uno de sus tests realizados

Este es un ejemplo de los tests realizados, como se puede apreciar, con los tests realizados se comprueban diferentes flujos, desde cuando el objeto pasado tiene únicamente un atributo, hasta cuando tienen varios.
La salida que proporciona Jest cuando se pasan los tests es bastante intuitiva y permite la rápida localización de los fallos (ver Figura 19).

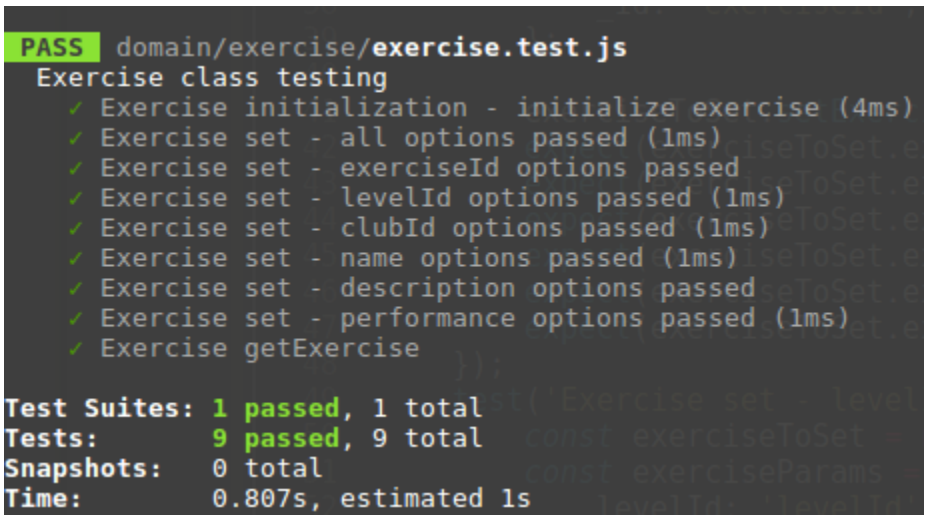


Figura 19: Salida de los tests del módulo Exercise

Con estas pruebas he podido comprobar lo ventajosos que son los tests en algunas ocasiones dado que, en alguna otra ocasión, el cambiar de una clase a otra, introduce errores en el código, que en primera instancia no son fáciles de detectar. Incluso con jshint ha habido ocasiones en las que los errores detectados por este, no eran suficientes.

5.1.3 Pruebas de usuarios

Con el fin de tener mayor claridad al respecto de cómo se ha concluido la aplicación, se han realizado pruebas con un grupo reducido de potenciales usuarios, miembros de un club deportivo ya existente.

Antes de realizar estas pruebas, se comprobó mediante la aplicación GetIt mencionada en el capítulo 3, las diferentes rutas del servidor, comprobando así que no devolvieran información que alguno de los usuarios no debiera ver (por su rol o por pertenecer a otro club).

Sobre estas pruebas, a pesar de haberse podido aplicar de forma más extensa (en tiempo y datos), se ha preferido limitarla, debido a que la información que se suele manejar es principalmente de menores, y por seguridad era preferible que solo estuviera en el sistema la información de aquellos usuarios que hubieran dado su consentimiento previo.

En cuanto a los resultados de esta, resulta interesante comprobar que tanto administradores, entrenadores, deportistas y tutores coinciden en que es una herramienta que no tiene demasiadas funciones inconsistentes, no hacía falta aprender demasiado para usarla, a pesar de que no se sintieran totalmente seguros al usarla por primera vez. También coinciden en que se puede aprender a usar muy rápidamente, aunque ante la pregunta de si es fácil de usar ha mostrado, que no es totalmente fácil de usar, pero de hecho en un caso han puesto en los comentarios de la encuesta que con una explicación de dos líneas se puede usar perfectamente (con el rol de administrador).

La principal diferencia destacada entre los diferentes roles ha sido que los administradores y entrenadores han encontrado la aplicación un poco más compleja que los deportistas y tutores. Si tenemos en cuenta que estos roles disponen de más funcionalidad dentro de la aplicación, se puede considerar aceptable esa disparidad.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Para concluir con este Trabajo de Final de Grado, vamos a hacer referencia a la Tabla 1 mostrada en el apartado del estado del arte a fin de posicionar la herramienta implementada frente al resto de las analizadas en el capítulo 2 referente al estado del arte.

	Diseño responsive	**Soporte deportes individuales**	**Soporte deportes colectivos**	**Intuitiva, o no sobrecargada**	**Gestión de usuarios**
GESDEPNET	No	No	Si	No	Si
sporteasy	No	No	Si	Si	Si (versión de pago)
Conjunto de aplicaciones Club TeamStuff	No	No	Si	Si	Si
Sport Club	Si	Si	Si	Si	Si

Tabla 12: Comparativa entre las aplicaciones explicadas en el estado del arte y la aplicación creada

Viendo la Tabla 12 y el desarrollo del proyecto, se aprecia que se han cumplido los requisitos deseados, los casos de uso, y que además se ha conseguido una aplicación que proporciona utilidades que no se encuentran disponibles en otras aplicaciones web actualmente, o por lo menos, que no son públicas para usuarios externos.
Además, se han realizado las pruebas necesarias para comprobar el correcto funcionamiento de cada una de las funcionalidades.
A nivel personal, con este Trabajo de Fin de Grado, he podido apreciar todo el trabajo que supone crear una aplicación web con su funcionalidad y complejidad al querer definir la arquitectura y querer al definir los diferentes modelos, tanto de la base de datos como al cómo definir los componentes.

6.2 Trabajo futuro

Existen amplias posibilidades de trabajo, futuro, dado que el deporte tiene en sí mismo infinitud de posibilidades, por ejemplo, quizá para los deportes paralímpicos sea necesario el estudio de cómo adaptar la interfaz para que sea accesible para todos los deportistas. Por ejemplo, quizá tener la posibilidad de audio para aquellos que puedan tener dificultades de visión, pero que aun así puedan utilizar en mayor o menor medida un dispositivo electrónico.
Por otra parte, existe la posibilidad de internacionalizarlo, dando soporte a múltiples idiomas. Incluso se podría añadir la posibilidad de poseer un traductor interno para aquellos clubes que posean padres, tutores y/o deportistas extranjeros, puesto que a veces

pasa que los niños hablan el idioma local, pero los padres tienen dificultades para hablarlo, especialmente en los términos más técnicos. Añadir un traductor sería la solución para estos casos.
En cuanto a funcionalidad que sea posible cercanamente, sin necesidad de paquetes externos o estudios, sería la posibilidad de controlar el material del club. De forma que se tenga controlado en que entrenamientos y eventos se usa y en los casos en los que pueda ser prestado, quien lo ha usado, quien lo tiene y si lo ha devuelto.

Referencias

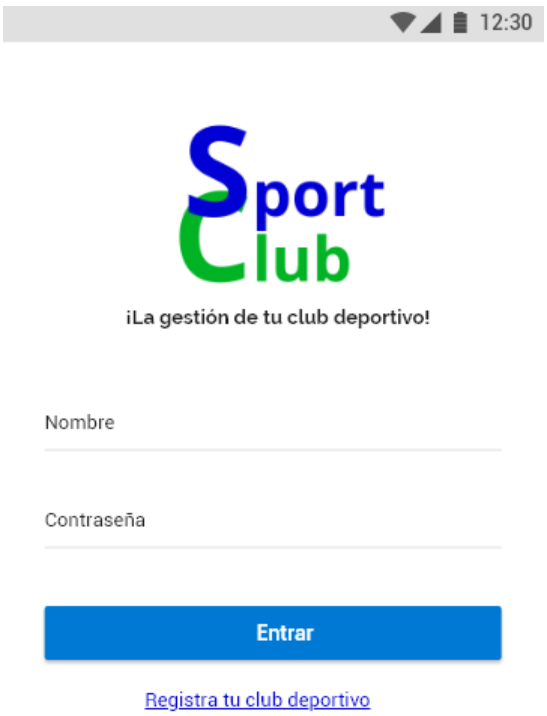
[1] GESDEP [consulta: junio 2019] Disponible en https://www.gestiondeportiva.com/web/
[2] SportEasy [consulta: junio 2019] Disponible en: https://www.sporteasy.net/es/home/
[3] TeamStuff [consulta: junio 2019] Disponible en: https://teamstuff.com/
[4] Encuesta StackOverFlow [consulta: junio 2019] Disponible en: https://insights.stackoverflow.com/survey/2019/#most-popular-technologies
[5] Angular [consulta: junio 2019] Disponible en: https://angular.io/docs
[6] IBM Cloud Education. *MEAN stack*, 9 de mayo de 2019 [consulta: junio 2019] Disponible en: https://www.ibm.com/cloud/learn/mean-stack-explained
[7] Ecma International, Standard ECMA-262 6th Edition, ECMAScript 2015 Language Specification (2015) [consulta: junio 2019] Disponible en: https://www.ecma-international.org/ecma-262/6.0/
[8] Can I Use ES6 [consulta: junio 2019] Disponible en https://caniuse.com/#search=es6
[9] Babeljs [consulta: junio 2019] Disponible en https://babeljs.io/docs/en/
[10] Bower [consulta: junio 2019] Disponible en https://www.npmjs.com/package/bower
[11] NPM [consulta: junio 2019] Disponible en https://www.npmjs.com/about
[12] NPM version [consulta: junio 2019] Disponible en https://docs.npmjs.com/cli/version
[13] MongoDB [consulta: junio 2019] Disponible en https://www.mongodb.com/json-and-bson
[14] NUÑEZ, Javier. *Express, un framework para nodejs* [consulta: junio 2019] Disponible en https://www.solucionex.com/blog/expressjs-un-framework-para-nodejs
[15] Angularjs [consulta:junio 2019] Disponible en https://github.com/angular/angular.js
[16] *¿Qué es Nodejs?*, Noviembre 2011 [consulta:junio 2019] Disponible en https://www.nodehispano.com/2011/11/que-es-node-js-nodejs/
[17] MomentJs [consulta:junio 2019] Disponible en https://momentjs.com/docs/
[18] AngularJs Material [consulta:junio 2019] Disponible en https://material.angularjs.org/latest/
[19] Jest [consulta:junio 2019] Disponible en https://jestjs.io/
[20] RANJIT, Preethi. *Guide to Linting JavaScript with JSHint* Actualizado: 24 de abril de 2019 [consulta: junio 2019] Disponible en https://www.hongkiat.com/blog/code-optimisation-linting-jshint/
[21] JSHint [consulta: junio 2019] Disponible en https://jshint.com/docs/
[22] Sublime Text [consulta: junio 2019] Disponible en https://www.sublimetext.com/
[23] GetIt [consulta: junio 2019] Disponible en https://getit.bartkessels.net
[24] Morgan [consulta: junio 2019] Disponible en https://www.npmjs.com/package/morgan
[25] Winston [consulta: junio 2019] Disponible en https://github.com/winstonjs/winston
[26] CODE Q&A: *¿Qué hace body-parser con express?* [consulta: junio 2019] Disponible en https://code.i-harness.com/es/q/2488309

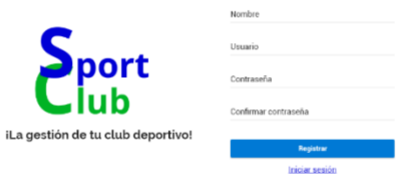
Glosario

SPA Single Page Application
JSON JavaScript Object Notation
BSON Binary JSON
HTTP Hypertext Transfer Protocol
REST Representational state transfer

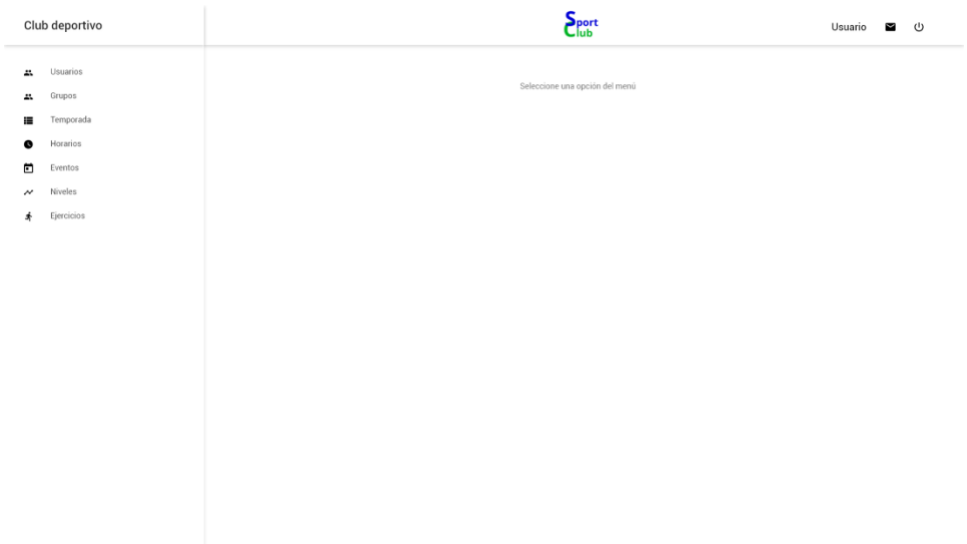
Anexos

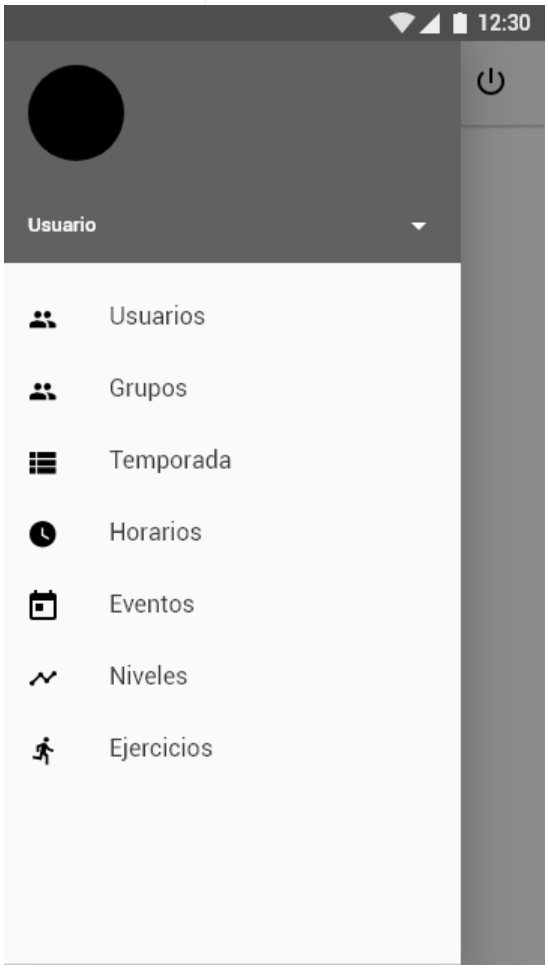
A Maquetas

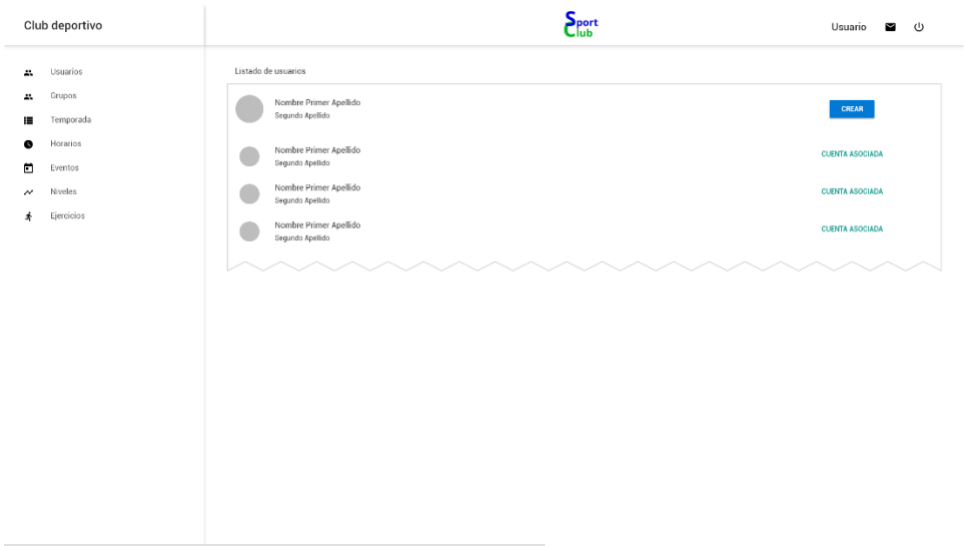


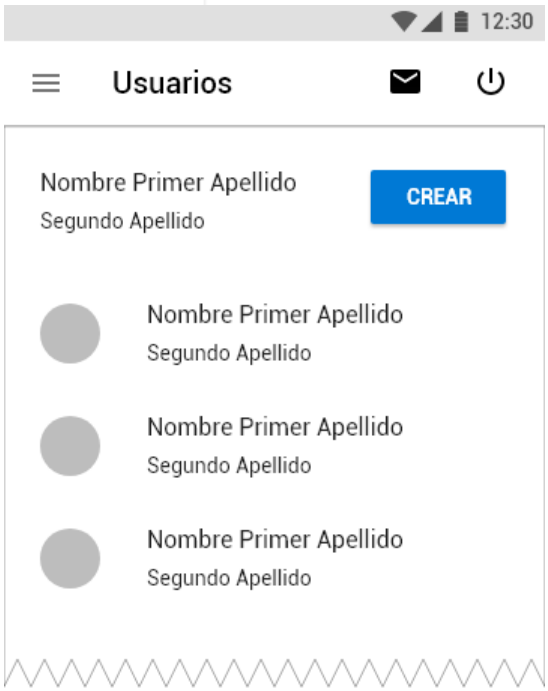

Sport
Club
¡La gestión de tu club deportivo!
Nombre
Usuario
Contraseña
Confirmar contraseña
Registrar
Iniciar sesión


12:30
Sport
Club
¡La gestión de tu club deportivo!
Nombre
Usuario
Contraseña
Confirmar contraseña
Registrar
Iniciar sesión


Club deportivo
Usuario
Usuarios
Grupos
Temporada
Horarios
Eventos
Niveles
Ejercicios
Seleccione una opción del menú


12:30
Usuario
Usuarios
Grupos
Temporada
Horarios
Eventos
Niveles
Ejercicios


Club deportivo
Usuario
Usuarios
Grupos
Temporada
Horarios
Eventos
Niveles
Ejercicios
Listado de usuarios
Nombre Primer Apellido
Segundo Apellido
CREAR
CUENTA ASOCIADA


12:30
Usuarios
Nombre Primer Apellido
Segundo Apellido
CREAR
Nombre Primer Apellido
Segundo Apellido
Nombre Primer Apellido
Segundo Apellido
Nombre Primer Apellido
Segundo Apellido